NASA/TM–2009-215388

Data Parallel Line Relaxation (DPLR) Code User Manual Acadia - Version 4.01.1

Dr. Michael J. Wright
Ames Research Center
Moffett Field, California

Todd White and Nancy Mangini
ELORET Corporation
Ames Research Center
Moffett Field, California

October 2009

The NASA STI Program Office . . . in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA Scientific and Technical Information (STI) Program Office plays a key part in helping NASA maintain this important role.

The NASA STI Program Office is operated by Langley Research Center, the Lead Center for NASA's scientific and technical information. The NASA STI Program Office provides access to the NASA STI Database, the largest collection of aeronautical and space science STI in the world. The Program Office is also NASA's institutional mechanism for disseminating the results of its research and development activities. These results are published by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA's counterpart of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or cosponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services that complement the STI Program Office's diverse offerings include creating custom thesauri, building customized databases, organizing and publishing research results . . . even providing videos.

For more information about the NASA STI Program Office, see the following:

- Access the NASA STI Program Home Page at *http://www.sti.nasa.gov*

- E-mail your question via the Internet to help@sti.nasa.gov

- Fax your question to the NASA Access Help Desk at (301) 621-0134

- Telephone the NASA Access Help Desk at (301) 621-0390

- Write to:
  NASA Access Help Desk
  NASA Center for AeroSpace Information
  7115 Standard Drive
  Hanover, MD 21076-1320

NASA/TM–2009-215388

Data Parallel Line Relaxation (DPLR) Code User Manual Acadia - Version 4.01.1

Dr. Michael J. Wright
Ames Research Center
Moffett Field, California

Todd White and Nancy Mangini
ELORET Corporation
Ames Research Center
Moffett Field, California

National Aeronautics and
Space Administration

Ames Research Center
Moffett Field, California 94035-1000

October 2009

Acknowledgements

The following individuals have made major contributions to the development, functionality and documentation of the DPLR Code Package.

Michael Barnhardt (ELORET /ARC)	Dual-time Accuracy DES Turbulence Modeling Extensions
David Boger (ARL)	DPLR Overset Integration Dual-time Accuracy
Dr. James L. Brown (ARC)	Menter SST Turbulence Model
Dr. Matt MacLean (CUBRC)	Automatic Interface Detection Subsonic Inlet Boundary Conditions Spalart-Allmaras Turbulence Model
Nancy Mangini (ELORET / ARC)	User Manual
Ryan McDaniel (ARC)	Baldwin-Lomax Turbulence Model
Dr. Grant Palmer (ELORET / ARC)	Chapman Viscosity Model
Dr. Dave Saunders (ELORET /ARC)	In-line Grid Adaption Run-time Control Template Utility
Dr. Chun Tang (ARC)	User Training
Todd White (ELORET / ARC)	Project manager
Dr. Michael J. Wright (ARC)	Original author and creator

Available from:

NASA Center for AeroSpace Information
7115 Standard Drive
Hanover, MD 21076-1320
(301) 621-0390

National Technical Information Service
5285 Port Royal Road
Springfield, VA 22161
(703) 487-4650

Data Parallel Line Relaxation (DPLR) Code

Acadia - Version 4.01.1

User Manual

10/27/09

Contents

Contents

Chapter 1 – Overview

Contents

1.0 Introduction

Accurate predictions of the environment that a spacecraft will encounter when entering and passing through a flow field, such as the Earth or Mars atmosphere, at hypersonic speeds can be of enormous value to aeronautical designers. Such predictions become even more critical when the shape of the craft changes during a mission due to extreme surface ablation or unexpected damage. The ability to anticipate flow environment changes relative to new craft geometries adds vital data to the situation analyses that inform mission command decisions.

Data-Parallel Line Relaxation (DPLR) code is a computational fluid dynamic (CFD) solver that was developed at NASA Ames Research Center to help mission support teams generate high-value predictive solutions for hypersonic flow field problems in a minimum amount of time using readily available computational resources.

The DPLR Code Package is an MPI-based, parallel, full three-dimensional Navier-Stokes CFD solver with generalized models for finite-rate reaction kinetics, thermal and chemical non-equilibrium, accurate high-temperature transport coefficients, and ionized flow physics incorporated into the code. DPLR also includes a large selection of generalized realistic surface boundary conditions and "hooks" to enable efficient loose coupling with external thermal protection system (TPS) material response and shock layer radiation codes.

1.1 Acadia - DPLR Code Package Version 4.01.1

> ***Note**: Each release of the DPLR Code Package has an associated name and number, beginning with Acadia, Version 4.01.1.*

Using the DPLR Code Package to achieve predictive solutions for hypersonic flow field problems involves completion of five main tasks:

1. Creating a structured grid file.
2. Converting the grid file to a DPLR-readable format with FCONVERT.
3. Sending the converted grid file to a number of processors for parallel solution by DPLR2D or DPLR3D.
4. Extracting information from the solution needed to create a graphic or data presentation of results with POSTFLOW.
5. Creating graphic or data presentations of the predicted flow environment.

A more detailed description of each of these tasks is presented in Table 1 below.

Tasks 3, 4, and 5 (highlighted in Table 1) are completed using applications and software utilities distributed in the DPLR Code Package Version 4.01.1.

Table 1 Typical Sequence For Calculating Hypersonic Flow Environments with DPLR Code Package 4.01.1

Task	Description	Tool	Input	Output
1	Creating a structured, face-matching (and/or overset-type) grid that is likely to contain the shock wave created in the flow field when the object enters at a specified Mach number.	**GridGen** (or similar grid generation application)	Geometric coordinates for object under study	plot3D file (serial ASCII format)
2 (optional)	Processing plot3D grid file, reordered if necessary.	**SUGGAR**	*plot3D grid file from GridGen or similar grid generation application	plot3D grid file & domain connectivity (.dci) file (See Chapter 8)
3	Converting the plot3D grid file into a DPLR-readable format, breaking it first, if required by the problem, into a number of component blocks.	**FCONVERT**	* plot 3D file (either point-wise or overset) * Input file containing problem-specific information about the grid file being converted	XDR parallel file (for use by DPLR)
4	Processing converted grid file on a specified number of processors to work the problem in parallel, performing sufficient iterations of the calculations to reach a solution.	**DPLR2D or DPLR3D**	* FCONVERT XDR output file * Input file containing problem-specific information about the object under study, flow environment, and solution data requirements.	restart file (parallel XDR format)
5	Extracting information from the restart file needed to create graphic and/or numeric representations of the predicted flow environment.	**POSTFLOW**	* restart file * Input file containing specifications of the data required by other applications for solution presentation(s) or further post-processing	plot3D and/or data file(s)
6	Creating visual representations or data report(s) of solution for use in further problem analyses and outcome presentations.	**Tecplot** (or similar graphics application)	* data file and/or * plot3D file	Graphic representation of predicted flow environment

1.2 How to Use This Manual

This manual is intended to be both a user guide and a reference resource.

If you are new to DPLR, begin by reading Chapters 3, 4, and 5. This will give you the information you need to understand the basic functions and elements of the Code Package. Next, study Chapter 7 to gain insight on how to use what you have learned to run an actual simulation.

As your command of the software grows, consult Chapters 6, 8 and 9 to deepen your understanding of the files types, utilities, and detailed capabilities of the DPLR Code Package Version 4.01.1.

If you are already using an earlier version of DPLR, you may want to begin by reading the Release Notes in Section 1.3 below to learn about new features or changed elements available in the current version of the code.

1.3 Release Notes

A summary of the new features and code changes implemented since the previous baslined release – in this case DPLR Code 4.01.0 – can be found in two locations:

- RELEASENOTES file in the main folder of the code distribution
- Section 1.4 below

1.4 New Features

The Acadia version of the DPLR Code Package contains five major new features:

- Overset Grid Capability
- Enhanced Time Accurate and Statistics Cabability
- Improved Turbulence Models
- Pointwise Surface and Integrated Aerodynamic Data Extraction
- Improved Support of Blowing Wall Boundary Conditions

1.4.1 Overset Grid Capability

DPLR now supports mixed face-matched and overset grid topologies. This capability is disabled by default, but can be enabled when the code is compiled if both DiRTLib and P3Dlib are present. Overset function is controlled in the Overset Grid Implementation section of the DPLR input deck, and requires the domain connectivity file generated by SUGGAR in a pre-processing step.

Chapter 8 in this manual contains a complete discussion of the Overset Capability currently available in DPLR.

1.4.2 Enhanced Time Accurate and Statistics Capabilities

DPLR now supports dual-time stepping in the Time Accurate & Statistical Options section of the DPLR input deck, and POSTFLOW can now post-process mean and root mean square (RMS) values in addition to the standard instantaneous values.

> ***Note**: The calculated mean and RMS values are exact for primitives, and approximate for other derived flow or surface quantities.*

1.4.3 New and Improved Turbulence Models

The Acadia release of DPLR contains improvements in three non-algebraic turbulence models:

- **Menter SST models.** The Menter SST models (`itmod` = 200X) now controls omega throughout viscous sublayers more effectively by attempting to merge omega with the analytical solution rather than just at the first point, thereby making SST less affected by grid density. A new Menter SST model (itmod=2004) includes additional compressibility corrections and corrections from Coakley, and Catris & Apoix.

- **DES extensions.** Detatched Eddy Simulations are hybrid models combining RANS (Reynolds Averaged Navier-Stokes) for near-wall modeling with an LES-like capability (Large Eddy Simulation) in regions where the flow becomes highly separated. DES extensions are now available for Menter SST and Spallart-Almars models through the `itmod` flag in the DPLR Input Deck.

- **New Turbulence Models.** The Wilcox 2006 model, preliminary versions of the Lag turbulence model, and an SST implementation similar to that in the NASA OVERFLOW model (called OSS internally) are now available to users for testing, although final implementation is still being evaluated.

1.4.4 Pointwise Surface and Integrated Aerodynamic Data Extraction

DPLR can now extract surface data (e.g. pressure, temperature, heat flux, and shear stress) at specified surface points for each iteration of the simulation, giving users the ability to display transient data for simulations of unsteady flows.

To use this new feature, locations of the specified points need to be listed in an ASCII file (`points.list`) that is placed in the working directory before beginning the simulation. DPLR reads this file only once, at the start of the simulation, extracts the data at the nearest surface point to each of the x, y, and z coordinates in the list, then places the results of these extractions back into the working directory as a series of files named `point.point#` where the `point#` is determined by the point order specified in the `points.list` file.

In addition to pointwise surface data, DPLR 4.01.1 can also extract integrated aerodynamic variables for each iteration of the simulation to track progress over time for body forces and moments. DPLR places the results of these extractions in a file named `aero.dat` in the working directory. This capability is enabled by the `iaero` flag in the Time Accurate and Statistical Options section of the DPLR input deck.

1.4.5 Improved Support of Blowing Wall Boundary Conditions

DPLR now supports blowing wall boundary conditions for any and all species listed in the simulation chemistry file, and is limited only by the stability of the chemistry model employed.

> ***Note**: This "new feature" corrects a problem in the code that previously limited DPLR to supporting blowing wall boundary conditions for only the first species listed in the chemistry file.*

Chapter 2 - Installation Guide

Contents

2.0 Introduction

The DPLR Code has been designed to achieve optimal performance on distributed memory parallel machines, making the code widely portable to a variety of architectures, from laptops, networked desktop workstations, and simple LINUX clusters to dedicated supercomputers.

2.1 System Requirements

The DPLR Code has been successfully installed and run on the following hardware / system software configurations:

Table 2.1 - Supported Hardware/Software Architectures

Architecture	Compiler	MPI Version
Xeon 32 bit	Intel, Portland, Lahey	MPICH, LAM-MPI, MPICH
Xeon 64 bit	Intel, Portland	MPICH, LAM-MPI
Xeon Dual Core	Intel	MPICH
Opteron 64 bit	Intel, Portland	MPICH, LAM-MPI
Intel (Mac) 32 bit	gfortran	MPICH
Altix	Intel	Open-MPI

Each of these architectures can be specified with the configuration script 'config' during the installation process described in Section 2.3, although on new systems some editing or creation of the `makefile.comm` and `include/machine.h` may be necessary.

2.2 Software

Two software packages must be installed before DPLR Code can be installed:

- Fortran 90 - DPLR is written entirely in Fortran 90 running on a UNIX/LINUX operating system and thus requires a working f90 compiler on the destination machine.
- Message Passing Interface (MPI) – DPLR Code uses MPI calls to facilitate inter-processor communications, so an MPI library must be present in the system.

Although not strictly required for successfully installing DPLR Code, having the following software packages on your system will enhance the utility and/or performance of the DPLR Code Package:

- FXDR – *fxdr* libraries provide a Fortran-based interface to the native XDR (external Data Reference) calls on all UNIX/LINUX machines. XDR enables the creation of platform-independent binary files, greatly enhancing the portability of generated datasets (e.g. restart and grid files). The code can be compiled without the *fxdr* libraries, however in that case, all restart and grid files must be written in either ASCII or machine-specific native binary format.[1]

- TECIO.A – These I/O libraries are used by POSTFLOW to create Tecplot binary data files for post-processing output.[2] Currently, DPLR can use Tecplot 360 or Tecplot II libraries. However, ASCII data in Tecplot can always be written by POSTFLOW, regardless of TECIO.A availability.

- LIBGOTO (Basic Linear Algebra Subroutines (BLAS) routines) - DPLR makes use of several BLAS routines for matrix-vector and matrix-matrix manipulations. Having such libraries on the target machine will result in a 20-25% performance improvement in the overall runtime of DPLR Code.[3]

[1] fxdr libraries are freely available at http://meteora.ucsd.edu/~pierce/fxdr_home_page.html

[2] Tecplot® I/O libraries are included with Amtec's Tecplot® visualization software, and may be available for free at http://www.tecplot.com/

[3] BLAS libraries are generally available from compiler makers as a part of their mathematical libraries for a nominal fee. In addition, several freeware sources exist. In particular, for Pentium or AMD architectures a freeware distribution called *libgoto* is available from http://www.tacc.utexas.edu/resources/software/

2.3 Installing the DPLR Code Package

The 4.01.1 version release of the DPLR Code Package consists of two gzipped tar files designated as follows:

- dpcodeV4.01.1.tar.gz – containing four separate executables
- samplesV4.01.1.tar.gz – containing a set of sample problems, complete with grids, input decks, and running instructions.

Step 1: **Unzip** the DPLR Code file.

Action: At the command line prompt, type:

```
gunzip dpcodeV4-01-1.tar.gz
```

Result: The archived file is renamed `dpcodeV4-01-1.tar`

Step 2: **Untar** the DPLR Code file.

Action: At the command line prompt, type:

```
tar -xvf dpcodeV4-01-1.tar
```

Result: A directory structure is created. (See Section 2.4 for more information on directories and files.)

Step 3: **Run** the *config* script.

Action: At the command line prompt, type:

```
./config
```

Result: If you are attempting to compile on a system the config script can recognize, a `makefile.comm` file is generated containing machine-specific information for your system. Otherwise, you will need to modify such a file yourself. Samples based on known MPI and FORTRAN builds can be found in the `defs` directory. After extracting the archive, a blank `makefile.comm` is created that includes descriptions of each of the necessary compiler options and paths.

Step 4: **Create** executables files.

Action: At the command line prompt, type:

```
make
```

Result: Assuming there were no problems, all executable files in the package are created. Links to executables `dplr2d, dplr3d, fconvert, postflow` are located in the `bin` directory.

2.4 Directory / File Contents

The directories and files resulting from untarring the DPLR Code Package contain the following components:

`bin/` – links to compiled binaries

`cfdinput/` – physical modeling data files used by DPLR during execution

`cfdlib/` – subroutines common to DPLR2D and DPLR3D

`config*` – a configuration script used to set up the makefile for the specific machine architecture

`defs/` – makefile templates for supported machines

`dplib/` – subroutines common to the entire package

`docs/` – information about DPLR Code Package documentation

`dplr2d/` – subroutines unique to the DPLR2D code

`dplr3d/` – subroutines unique to the DPLR3D code

`fconvert/` – subroutines unique to the FCONVERT code

`include/` – modules, common blocks and other include files that are incorporated into the various executables

`makefile` – makefile for the package

`makefile.comm` – compiler, architecture and library options for building DPLR (may be created by `config` script)

`post/` – subroutines unique to the POSTFLOW code

`utilities/` – utility codes and scripts distributed with the package

Tech Tip: *The contents of each directory distributed with the DPLR Code Package are required for that version of DPLR to function properly. Thus, removing files from any of the directories is not recommended.*

Chapter 3 – Using FCONVERT

Contents

3.0 Introduction

The primary function of FCONVERT is to read `plot3d` grid files generated by third-party software applications (such as GridGen) and convert them into a format that can be used by DPLR to solve hypersonic CFD problems.

However, you can also use FCONVERT to change the format of a restart file, convert a `plot3d` flow file (a.k.a. function file) into a restart file, process input radiation and boundary condition files, and change the number of processors on which a simulation can be run.

Finally, you can use FCONVERT to manipulate a `plot3d` grid file through scaling it by a multiplicative factor (useful when changing grid units, e.g., from feet to meters) and to keep the DPLR solution computationally efficient by:

- "sequencing" or coarsening the grid in one or more dimensions.
- "breaking" or decomposing the grid into multiple pieces to run on a parallel machine. [1]

3.1 Running FCONVERT

Step 1: **Open** the text editor program for your system.

Action: At the command line prompt, type:

/[path to your fconvert directory]/`file_convert.inp`

Result: An input file or "deck" appears on screen with place-holder default values. To start with a blank deck, delete the default values as shown on the following page.

[1] Unlike DPLR2D and DPLR3D, FCONVERT is a serial code, so all pre-processing must be done on a single processor.

```
      Input file for fconvert

iaction ifile   idim   iinfo   ivers   nvers

inform  inint  idummy  nborig

ouform  ouint  odummy  ncedge

imseq   iscale   sfact  imir

nbreak

Decomposition information for each master block
ibrk    jbrk    kbrk

Sequencing information for each master block
iseq    jseq    kseq

iname,xname,cname

oname

nsin  nerin  nevin  necin  ntbin

imirx  imiry  imirz
```

Figure 3-1 FCONVERT Input Deck

Tech Tip: *Although you can add as many sections as you need to specify the decomposition and sequencing instructions for each master block in your input grid, take special care to preserve the line spacing within each block-specific section and throughout the global areas of the input deck as you enter new values and/or replace default values with problem-specific ones. If lines are added to or subtracted within these areas, DPLR will not be able to read the file accurately.*

Step 2: **Enter** values for each of the input variables or "flags". (See Section 3.2 for a description of input flags and a list of allowable values.)

Action: For each flag, type:

```
allowable, problem-specific value
```

Result: Input deck contains sufficient information for FCONVERT to process the input grid file and convert it into a DPLR-readable file.

Step 3: **Save** the file with your problem-specific name to your working directory.

Step 4: **Run** FCONVERT.

Action: At the command line prompt, type:

```
fconvert < yourinputdeckfilename.inp
```

Result: An output grid file in the format you specified through the `ouform` flag (usually XDR parallel) is created along with on-screen summary of actions performed by FCONVERT. (See Section 3.3 for an example of a problem-specific FCONVERT input deck and the output summary generated after running the program.)

3.2 Input Flags for FCONVERT

Input variables for FCONVERT are discussed below in the order they appear in the deck.

`iaction` – Specifies the action to perform. Allowable values are:

0	test decompose over a range of blocks
1	decompose file according to `(ijk)brk`
2	decompose file according to `nbreak`
3	recompose file into original blocks
10	format conversion only, no parallel decomposition or recomposition (scaling or sequencing still allowed)
11	stop after printing file size (determine the dimensions and number of computational cells in each block and output this information to screen)

`ifile` – Specifies the type of file to be processed. Allowable values are:

1	grid file
2	restart / flow file
3	boundary condition (BC) file
4	radiation file

***Tech Tip**: Because restart, boundary condition, and radiation files are never decomposed or recomposed, the only functional actions for these file types are* `iaction` *= 10 (format conversion) and* `iaction` *= 11 (file size).*

`idim` – Specifies the dimension of the input file. Allowable values are:

2 2D/Axisymmetric
3 3D

***Tech Tip**: Whatever value* `idim` *is set to, the input grid file must match it (e.g., if* `idim=2`*, the input grid file must be two-dimensional; if* `idim=3`*, the input grid file must be three-dimensional. However, some grid generation programs, like GridGen, do not support 2D grid generation. In this case, you need FCONVERT to "see" a 3D grid file as a 2D file. To accomplish this, make sure that the k-dimension =1 in the input grid file. FCONVERT will automatically strip the z-coordinates from a 3D input grid file if the k-dimension =1, regardless of the* `idim` *setting. For all other cases, FCONVERT will execute the operation if given an input grid file of different dimension than specified by the* `idim` *flag, but the results may be undesirable.*

`iinfo` – Controls the output of debugging information. Allowable values are:

0 Do not output debugging information
1 Output debugging information

***Tech Tip**: This information is intended for software developers.*

`ivers` – Specifies the DPLR Code version of the output file. Allowable values are:

1 Do not attempt to change file version
2 Upgrade file to the current version of DPLR Code
3 Convert file to the DPLR Code version specified by `nvers`

***Tech Tip**: All parallel grid, restart, boundary condition, and radiation files are DPLR Code Version-specific. With the exception of boundary condition files, `ivers` allows the format conversion of all supported files between all release versions of the DPLR Code Package.*

`nvers` – Specifies the DPLR Version to convert the output file to when `ivers` = 3. Allowable values are the real numbers of the major and minor releases of the DPLR Code Package, from 2.31 through 4.01.1.

`inform` – Specifies the format of input file. (See Appendix A for more information about supported I/O formats.) Allowable values are:

1	Unformatted parallel file
2	Unformatted plot3d (grid or q) file
3	Unformatted plot3d (grid or function) file
11	XDR parallel file
21	ASCII parallel file (used for debugging)
22	ASCII plot3d (grid or q) file (used for debugging)
23	ASCII plot3d (grid or function) file (used for debugging)

***Tech Tip:** Because DPLR is a double precision code, be sure that all grid files being imported were also created as double precision.*

`inint` – Specifies whether an input zonal interface file is to be read by FCONVERT or whether zonal interfaces are to be computed automatically. Allowable values are:

0	Do not read input interface file
1	Read input interface file
2	Auto-detect full face interfaces ONLY
3	Auto-detect all interfaces (fast method)
4	Auto-detect all interfaces (accurate method)

Tech Tip: *Input zonal interface information is only required when the input file is a serial plot3D file. The zonal interface file required in this instance can either be created by hand (`inint=1`) or can be created automatically by FCONVERT which will then embed it into the DPLR-readable grid file it creates (`inint=2-4`). Because the automatic creation of zonal interface files may require a substantial amount of time and computing resources, you can tell FCONVERT to also write the information it generates to a separate file by setting the `ouint flag>0`, thus eliminating the need to regenerate the information every time the problem is run. See Section 3.4 for more information on zonal interface files.*

`idummy` – Specifies whether or not the input grid file contains dummy (a.k.a. "ghost") cells. Allowable values are:

0 Input file does not contain dummy cells

1 Input file contains dummy cells

Tech Tip: *Because DPLR automatically generates grid dummy cells as necessary at runtime, considering their presence during grid generation is usually unnecessary and* **`idummy`** *typically remains = 0.* **`idummy`** *should only =1 if:*

* *you choose to generate your own grid dummy cell coordinates rather than allowing DPLR to do it*

or

* *you are converting a function file into a restart file*

Note, Input dummy cells will be discarded if mesh sequencing is enabled (`imseq = 1`).

`nborig` – Specifies the numbers of master blocks in the file. Allowable values are:

- the actual number of blocks in the input file

or

- the final number of blocks after a grid file recomposition (`iaction=3`) has been performed.

ouform – Specifies the format of the output file. (See Section 9.2 for more information about supported I/O formats.) Allowable values are:

0	Do not generate an output file (used for debugging)
1	Unformatted parallel file
2	Unformatted plot3d (grid or q) file
3	Unformatted plot3d (grid or function) file
11	XDR parallel file – preferred for file read into DPLR
21	ASCII parallel file (used for debugging)
22	ASCII plot3d (grid or q) file (used for debugging)
23	ASCII plot3d (grid or function) file (used for debugging)

ouint – Specifies whether an output zonal interface file is to be written and saved for future use. Allowable values are:

0	Do not write output interface file
1	Write output interface file
2	Write output interface file including dummy cells (used for debugging)
11	Write output interface file including edges (used for debugging)
12	Write output interface file including dummy cells and edges (used for debugging)

Tech Tip: *When* `inint`*=2-4, FCONVERT automatically creates zonal input information for the plot3D grid file being processed and embeds the information into the file being produced. By setting* `ouint` *>0, you tell FCONVERT to also write the zonal interface information it creates to a separate file, thus eliminating the need to regenerate it – a potentially time- and resource-consuming task - if the problem is re-run in the future. Note, however, that if you change the grid topology in any future run of the problem, the zonal interface files will need to be recalculated.*

odummy – Specifies when an output file contains dummy cells. Allowable values are:

0 Output file does not contain dummy cells

1 Output file contains dummy cells (use for debugging)

Tech Tip: *The appropriate setting of* `odummy` *is nearly always 0.*

`ncedge` – Specifies which edge and corner interfaces should be generated. Allowable values are:

0 Do not compute and edge and corner interfaces

1 Compute all edge and corner interfaces

2 Compute only edge/corner interfaces created by decomposition

Tech Tip: *This flag should always be set to 1, unless used for software development.*

`imseq` – Specifies whether mesh sequencing (or coarsening) is to be performed and in which computational direction(s). (See Section 3.6 for more information on mesh sequencing.) Allowable values are:

0 Do not sequence the file

1 Sequence according to the values of `(ijk)seq`

2 Sequence all blocks using `(ijk)seq` values

-2 Upsequence a restart file

`iscale` – Instructs FCONVERT to scale an input grid file (`ifile`=1) by a constant multiplicative factor (`sfact`) before creating a DPLR-readable output grid file. Allowable values are:

0 Do not scale input grid file

1 Scale input grid file by `sfact`

Tech Tips:

1) *This option is typically used to convert grids to SI units.*

2) *If* `iscale` *is set to 1 for any file type other than a grid file, FCONVERT will silently reset it to zero.*

sfact – Specifies the multiplicative scale factor to use when `iscale`=1.

imir – Specifies whether to mirror the input grid or restart file across one or more axes. Allowable values are:

0 Do not mirror input file

1 Mirror input file according to `imir(xyz)`

2 Copy and mirror input file according to `imir(xyz)`

Tech Tip: *`imir` is used primarily to generate a reflected grid or restart file in preparation for starting a full body simulation. If `imir` is set to 1, valid entries must be specified for `imirx, imiry`, and `imirz` (mirroring factors across the yz, xz, and xy axes). When mirroring is turned on, FCONVERT will mirror the appropriate xyz (or uvw) variable on output, automatically reverse the order of the grid in the i-direction in each block if necessary to ensure that the output grids and solutions files remain right-handed, and determine the new zonal-interface definitions that result from this mirroring. However, you must manually correct the boundary conditions in the DPLR input deck by reversing the BC numbers of the `imin` and `imax` faces in each block! And when using FCONVERT to create or mirror a restart file from an input file other than DPLR's pslx format, be sure to set correct values for `nsin`, `nerin`, `nevin`, `necin`, and `ntbin` so that FCONVERT can correctly determine the location of the velocity components in the file.*

If `imir` is set to 2, FCONVERT will consult the valid entries specified for `imirx, imiry`, and `imirz`, retain the current grid, copy it, and add a mirrored version to the new file.

---

nbreak – Specifies the number of blocks to decompose the input file into when `iaction` =2.

ibrk,jbrk,kbrk – Specifies grid decomposition factors in the *i-*,*j-*,and *k*-directions for each master block when `iaction`=1.

Tech Tip: *The most common use of FCONVERT is to decompose an input grid file into blocks for simultaneous, parallel execution on a number of processors. When the number of processors to be used for the solution run has been determined, the input grid file must be decomposed into at least one block per processor. This can be accomplished in two ways:*

** Set* `iaction`*=1, manually determine the best strategy for decomposing the input grid file into master blocks, then enter one set of decomposition factors (*`ibrk`*,* `jbrk`*, and* `kbrk`*) for each block in the input file.*

or

** Set* `iaction`*=2, enter the number of blocks (minimally equal to the number of available processors for the run) to decompose the input file into in* `nbreak`*, then allow FCONVERT to automatically determine the best decomposition strategy for the input grid file. Although this method may produce a good result from a CPU load balance perspective, it may not produce a good flow-solver result.*

Note: When parallel decomposition is not being performed, setting `ibrk=-1` *on the first line tells FCONVERT not to read additional block decomposition records.*

`iseq,jseq,kseq` **–** Specifies sequencing factors in the *i-*,*j-*,and *k*-directions when `imseq=1,2,`or`-2.`

Tech Tip: *One set of sequencing factors is required for each block in the input file unless* `imseq=2` *or* `-2.` *These settings tell FCONVERT to sequence (remove points) or upsequence (add back points) all grid blocks by the same factor so only one set of sequencing factors are required regardless of the number of grid blocks.*
Note: When sequcing is not being performed, i.e., `imseq=0`*, setting* `iseq=-1` *on the first line tells FCONVERT not to read additional block sequencing records.*

iname – Specifies the input file name. This is the file that will be processed by FCONVERT. The filename should be surrounded by single or double quotes, and can be specified with either a relative or an absolute path as shown in the example below:

```
'./ASCIIPlot3Dfilename.g'
```

Tech Tip: *The suffix used in the file name is optional. FCONVERT will assume the default suffix for the specified file type if not manually entered. See Appendix A for a list of file types and associated default suffixes.*

xname – Specifies the name of a previously prepared, input zonal interface file when `inint`=1. (See Section 3.4.1 for more information on previous preparation of a zonal interface file.) The filename should be surrounded by single or double quotes, and can be specified with either a relative or an absolute path as shown in the example below:

```
'./YourZonalInterfaceFileName.inter'
```

Tech Tip: *The suffix used in the file name is optional. FCONVERT will assume the default suffix for this file type is ".inter" if not entered.*

cname – Specifies the CFD input deck file name (if any). This file is only used to locate solid walls to assist in decomposing the input grid when `iaction`=2 and FCONVERT is attempting to automatically break the input grid into blocks for the best possible parallel solution. The filename should be surrounded by single or double quotes, and can be specified with either a relative or an absolute path as shown in the example below:

```
'./CFDfile.inp'
```

Tech Tip: *The suffix used in the file name is optional. FCONVERT will assume the default suffix for this file type is ".inp" if not entered.*

oname – Specifies the output file name. This is the DPLR-readable file that will be created by FCONVERT to be solved by DPLR. The filename should be surrounded by single or double quotes, and can be specified with either a relative or an absolute path as shown in the example below:

```
'./XDRParallelgridfilename.pgrx'
```

Tech Tip: *The suffix used in the file name is optional. FCONVERT will assume the default suffix for the file type specified in* `ouform` *if not manually entered. See Appendix A for a list of file types and associated default suffixes.*

Note: If the output filename (with suffix) is the same as the input filename, the input file will be overwritten- not a typically desired result. Also, if an output interface file is requested by setting `ouint=1`*, the suffix* `'.inter'` *will be appended to the prefix specified by* `oname`*.*

---

nsin – Specifies the number of chemical species to be considered in the CFD solution. This is only read if you are trying to create a restart file from an input file other than DPLR's pslx format so that FCONVERT can correctly determine the location of the velocity components in the file.

nerin – Specifies the number of unique rotational temperatures (energy conservation equations) to be considered in the CFD solution. This is only read if you are trying to create a restart file from an input file other than DPLR's pslx format so that FCONVERT can correctly determine the location of the velocity components in the file.

nevin – Specifies the number of unique vibrational temperatures (energy conservation equations) to be considered in the CFD solution. This is only read if you are trying to create a restart file from an input file other than DPLR's pslx format so that FCONVERT can correctly determine the location of the velocity components in the file.

necin – Specifies the number of unique electronic temperatures (energy conservation equations) to be considered in the CFD solution. This is only read if you are trying to create a restart file from an

input file other than DPLR's pslx format so that FCONVERT can correctly determine the location of the velocity components in the file.

`ntbin` – Specifies the number of turbulence variables to be considered in the CFD solution. This is only read if you are trying to create a restart file from an input file other than DPLR's pslx format so that FCONVERT can correctly determine the location of the velocity components in the file.

`imirx,imiry,imirz` – Specifies mirroring factors acrosss the *yz*-, *xz*-, and *xy*-axes to be used when `imir` = 1. Allowable values are:

-1 or 0	No mirroring will take place along this axis
1	Mirroring will take place along this axis

Tech Tip: *It is an error to set all three of these flags to 1.*

3.3 'Neptune' Sample Case

The sample case used throughout the DPLR Code User Manual to illustrate how the Code Package works describes a Neptune probe with an ellipsoidal body as shown in Figure 3-2. This case is an example of aerocapture, where drag from the atmosphere is used to decelerate the vehicle and bring it into orbit.

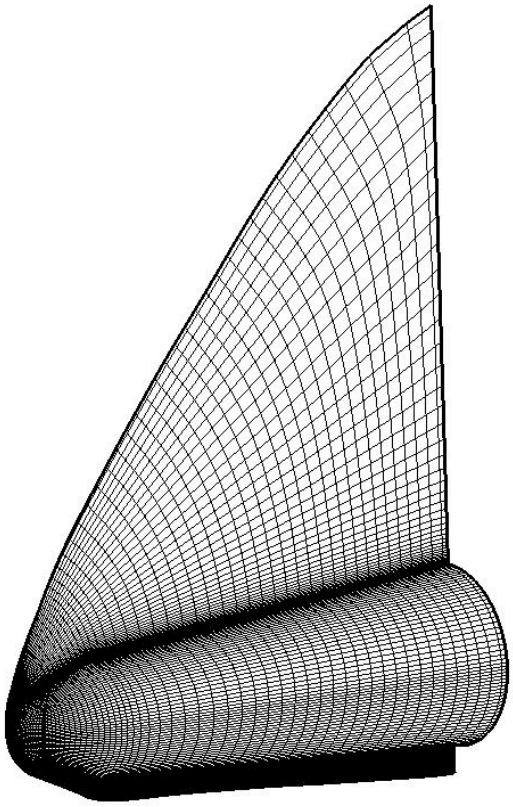


Figure 3-2 Neptune Probe

3.3.1 Neptune Input Deck

The input deck below shows the problem-specific entries to make for FCONVERT to process the serial plot3D grid file of this probe shown in Figure 3-2 into a DPLR-readable XDR parallel grid file.

```
Input file for fconvert

iaction ifile   idim   iinfo   ivers   nvers
  1       1       3      0       1     4.01.0

inform  inint  idummy  nborig
 22       1       0       2
```

```
ouform  ouint  odummy  ncedge
 11       0       0       1

imseq   iscale   sfact  imir
  0       0      1.0      0

nbreak
  1

Decomposition information for each master block
ibrk    jbrk    kbrk
  1       1       1
  7       1       1

Sequencing information for each master block
iseq    jseq    kseq
 1       1       1
 1       1       1

iname,xname,cname
'neptune'
'neptune'
'none'

oname
'neptune.8PE'

nsin  nerin  nevin  necin  ntbin
 5     0       1      0      0
```

Figure 3-3 FCONVERT Input Deck for Neptune Probe

3.3.2 Neptune Input Deck Settings

This is a three-dimensional problem. The input grid is ASCII plot-3D and the output grid file is parallel XDR. The original grid consists of two master blocks and must therefore include an interface file. (See Section 3.4 for more information on zonal interface files.) There are three interfaces between these two blocks. The following table explains the meaning of the input deck settings in this sample case.

Input Flag	Setting	Explanation
iaction	1	Break each master block the input grid along the *i, j,* and *k* axes as specified in ibrk, jbrk, kbrk for each block.
ifile	1	The input file is a grid file.
idim	3	The input file is a 3D file.
iinfo	0	Do not output debugging information.
ivers	1	Do not attempt to change file version.
nvers	4.01.0	Release version of the DPLR Code package being used. (Value ignored when ivers=1).
inform	22	Input file is an ASCII plot3D grid file.
inint	1	Read input interface file.
idummy	0	Input file does not contain dummy cells.
nborig	2	There are 2 master blocks in the input grid file.
ouform	11	The output file will be an XDR parallel grid file
ouint	0	Do not write an output interface file (one already exists!)
odummy	0	Output file does not contain dummy cells.
ncedge	1	Compute all edge and corner interfaces.
imseq	0	Do not sequence the input grid file.
iscale	0	Do not scale the input grid file.
sfact	1.0	Ignored value because iscale = 0
imir	0	Do not mirror input grid file.
nbreak	1	Ignored value because iaction = 1
ibrk, jbrk, kbrk	1, 1, 1, 7, 1, 1	Do not break the first master block in any direction. Break the second master block 7 times in the *I* direction only .
iseq, jseq, kseq	1, 1, 1 1, 1, 1	Values ignored (imseq=0).
iname	‘neptune’	The name of the input grid file is ‘neptune’.
xname	‘neptune’	The name of the input zonal interface file is ‘neptune’.

Input Flag (cont.)	Setting (cont.)	Explanation (cont.)
cname	'none'	When iaction=1, FCONVERT breaks master blocks according to the values in ibrk, jbrk, kbrk and ignores information in a CFD input deck file.
oname	'neptune-8PE'	The name of the output DPLR-readable grid file is 'neptune-8PE' – a file convention that notes how many processors were used to run the problem – in this case, 8.
nsin	5	Ignored value (not trying to create a restart file from a non-DPLR pslx input file)
nerin	0	Ignored value (not trying to create a restart file from a non-DPLR pslx input file)
nevin	1	Ignored value (not trying to create a restart file from a non-DPLR pslx input file)
necin	0	Ignored value (not trying to create a restart file from a non-DPLR pslx input file)
ntbin	0	Ignored value (not trying to create a restart file from a non-DPLR pslx input file)

3.3.3 Neptune Output Summary

When you run FCONVERT (See Section 3.1, Step 4), the program provides an on-screen summary of the actions performed along with some supplemental information as shown below.

```
 fconvert
 NASA Ames Version 4.01.0
 Maintained by Mike Wright;  last modified: 02/05/09
 ******************************************

 Reading plot3d asciifile  neptune.g
 Writing parallel XDR-formatted file  neptune-8PE.pgrx

 Input file does not include dummy cells
 Output file includes dummy cells

Input file is 3D

 Input Block  1 size: il = 32; jl = 16; kl = 64 (32768 cells)
 Input Block  2 size: il = 48; jl = 64; kl = 64 (196608 cells)

 Largest block is:
      nb = 2; original block = 2
      il = 48; jl = 64; kl = 64

 Read input interface file neptune.inter
 Found 3 valid zonal interface blocks in   2 block grid file

 Decomposing block   1 into  1: ibrk=  1 jbrk=  1 kbrk=  1
 Decomposing block   2 into  7: ibrk=  7 jbrk=  1 kbrk=  1
 ----------------------------------------
                  creating   8 total blocks

   8 Blocks;  Total load imbalance =  12.50%

 Output Block 1 size: il = 32; jl = 16; kl = 64 (32768 cells)
 Output Block 2 size: il =  7; jl = 64; kl = 64 (28672 cells)
 Output Block 3 size: il =  7; jl = 64; kl = 64 (28672 cells)
 Output Block 4 size: il =  7; jl = 64; kl = 64 (28672 cells)
 Output Block 5 size: il =  7; jl = 64; kl = 64 (28672 cells)
 Output Block 6 size: il =  7; jl = 64; kl = 64 (28672 cells)
 Output Block 7 size: il =  7; jl = 64; kl = 64 (28672 cells)
 Output Block 8 size: il =  6; jl = 64; kl = 64 (24576 cells)

 Largest block is:
      nb =   1; original block =    1
      il =   32; jl =   16; kl =   64
-----------------------------------------
 Summary (grid dimensions for CFD input deck):
 Hardwired to run on   8 processors

      Block   1; nx =  32; ny =  16; nz =  64
      Block   2; nx =  48; ny =  64; nz =  64

 ==> Finished writing output file neptune-8PE.pgrx
Done!
```

3.3.4 Neptune Output Summary Information

In addition to verifying the steps undertaken by FCONVERT, the output summary also provides specific information about how the master blocks from the input grid file were decomposed into a set of output blocks suitable for parallel processing by DPLR.

In this sample case, FCONVERT was told how to break the master blocks when `iaction` was set to = 1 and the `ibrk, jbrk, and kbrk` values for each master block were entered into the input deck (See Section 3.3.1.). These settings told FCONVERT to leave the first master block alone (to make 1 block) and break the second master block 7 times in the *i* direction (to make 7 blocks). This created a total of 8 blocks in the output XDR grid file and thus required or '`hardwired`' the file to be run on a minimum of 8 processors.

If `iaction` had been set to =2, the user would have had to know, in advance, how many processors their system could dedicate to running the problem and enter that value into `nbreak`. FCONVERT would then have calculated the best numerical solution for breaking the input grid into at least the number of blocks equal to the value in `nbreak` and displayed the resulting block dimensions and load imbalance information in the output summary.

See Section 3.5 for more information on parallel decomposition and load imbalance.

***Tech Tip**: Although FCONVERT-generated solutions for block decomposition are computationally accurate, they may not be the most practical way to handle grids for complex object geometries. Therefore, most DPLR users choose to keep `iaction=1` and determine from their own experience the best way to break the master blocks in the input grid, recognizing that there are algorithmic limits on how small parallel blocks should be, and thus determining how many processors the problem will require.*

3.4 Parallel Decomposition

DPLR is a distributed-memory parallel code, so solutions for each grid block are computed simultaneously rather than sequentially. Multi-block information transfer is handled through MPI data constructs, so it is necessary to run on *at least* as many processors as blocks in the original computational grid. Running on more processors than master grid blocks is often advantageous, since the largest blocks can then be split (decomposed) into smaller pieces, increasing computational efficiency and decreasing turnaround time. This decomposition, if required, is the most common reason for running FCONVERT.

Although the "ideal" number of processors to use for a given job is a matter of personal preference, it is generally a function of the total number of processors that are available and the number that are necessary to achieve a reasonable measure of computational efficiency referred to as "load balance".

Once the desired number of processors to use during the DPLR run has been determined, the plot 3D input grid file must then be decomposed into a minimum of one block per processor.

As discussed in Section 3.2, parallel decomposition of an input grid file can be accomplished in two ways:

- Setting `iaction=1`, manually determining the best strategy for decomposing the input grid file, then entering one set of decomposition factors in the `ibrk`, `jbrk`, and `kbrk` flags in the FCONVERT input deck for each block in the input file

  or

- Setting `iaction=2`, entering the number of blocks to decompose the input file into (minimally equal to the number of processors that will be used for the DPLR run) in the `nbreak` flag, and allowing FCONVERT to automatically determine the best decomposition strategy for the input grid file.

Although the choice of setting is dependent upon the situation, choosing `iaction=1` can have significant advantages, including:

- More direct control over the decomposition strategy to ensure minimal generation of additional zonal interfaces and to avoid breaks in the body-normal direction – two conditions that support the rapid convergence of DPLR Code solutions.
- Avoiding the need to generate the DPLR input deck prior to running FCONVERT (as is the case when `iaction=2.` See Chapter 6 for more information on this requirement.)

3.4.1 Load Balance

Load balance is a measure of the computational efficiency, and thus the operational quality, of a grid decomposition strategy.

Expressed in terms of *imbalance*, this metric is computed as the average amount of wasted CPU time that will result if the proposed grid decomposition strategy is employed to prepare the input grid for parallel processing.

Ideally, as the load imbalance value for a decomposition strategy approaches zero, the computational efficiency, and thus quality/desirability of the strategy increases.

In practice , however, things are often more complex and accepting a certain amount of load imbalance can be preferable to a decomposition strategy that introduces an

unacceptable number of zonal interfaces or one that requires block breaks in the body normal direction.

In most cases, using the load imbalance metric, estimated and reported by FCONVERT whenever a grid file is processed, is usually sufficient to provide a first-order estimate of the quality of a decomposition strategy.

***Tech Tip**: To test the load balance for a decomposition strategy before using it in a DPLR run, set* `iaction=0` *and* `nbreak` *= the maximum number of blocks in the strategy. FCONVERT will then output the most load balanced way to decompose the input grid into that number of output blocks.*

3.4.2 Parallel Recomposition

Although this action is rarely used, FCONVERT can be used to "recompose" a grid file that was previously decomposed by setting `iaction=3`, nborig= number of blocks in the recomposed file, and init=1.

***Tech Tip**: Although FCONVERT will recompose an input grid file, it does not recreate the zonal interface file for the recomposed file. Therefore, it is important to save the original interface file to avoid having to recreate it after the recompose is completed.*

3.5 Mesh Sequencing

Computational grids composed of a large number of data points typically take longer to solve than grids with fewer points. As a result, grids used for initial solutions of CFD problems are sometimes coarsened or "sequenced" to reduce the number of points while maintaining the topology of the mesh. After an acceptable "first guess" is acquired, the grid is restored in a step-wise fashion to its original number of points for final solution and post-solution data reporting.

Also, there may be a problem-specific advantage to obtaining a solution on a coarser mesh, as in the case of wake flow problems or for performing grid convergence studies.

For both these reasons, an option is included in FCONVERT to sequence (coarsen) a grid, radiation, boundary condition, or restart file, and create a new output file that maintains point-matching fidelity.

3.5.1 Sequencing an Input Grid

To sequence an input grid, set `imseq=1`, then enter a sequencing factor in `iseq, jseq`, and `kseq` for each master block in the grid. A sequencing factor of *n* implies that the block should be coarsened *n* times in that direction.

For example, a sequencing record of:

```
iseq    jseq    kseq
  3       2       1
```

tells FCONVERT to retain one out of every 3 points in the *i* direction, one out of every 2 points in the *j* direction, and every point in the *k* direction of the block being described.

To sequence every block in the grid by the same set of factors, set `imseq=2`, then enter only one set of sequencing factors in the `iseq, jseq`, and `kseq` input flags.

If the input grid has zonal interface information associated with it, these data will be automatically sequenced along with the grid file. Once you have appropriately sequenced the grid file, boundary condition file (if any), and radiation file (if any), you can set up and run the problem independently from the fine grid solution.

Tech Tip*: Be sure that the sequencing strategy you choose for multi-block problems results in a coarsened grid that remains point matched. While failure to produce a point-matched grid across zonal interfaces will not result in a runtime error in FCONVERT, it will cause problems in the DPLR run.*

3.5.2 Upsequencing Restart Files

After using a sequenced grid file to achieve a good initial solution in a relatively short period of computing time, you can proceed to restoring grid points and refining your solution by using FCONVERT to upsequence the restart file.

To upsequence the restart file generated with the coarsened grid:

Step 1: **Open and name** a new FCONVERT input file**.**

Step 2: **Set** `ifile=2`, `inform=11, imseq=-2, iseq, jseq, kseq` to values used during sequencing, `iname=` coarsened restart file name, `oname=` new (uncoarsened) restart file name (`*.pslx`).

Step 3: **Save** file to your working directory.

Step 4: **Run** FCONVERT < new FCONVERT input file.

Step 5: **Open and name** another new FCONVERT input file.

Step 6: **Set** `ifile=1`, `inform=2, imseq=0, iname= original plot3d grid filename, oname= new (unsequenced) XDR parallel grid file name (*.pgrx).`

Step 7: **Save** file to your working directory.

Step 8: **Run** FCONVERT < second new FCONVERT input file.

*Result***:** Your working directory now contains an upsequenced restart file that can be used to start a new solution run with the DPLR-readable grid file containing the original number of data points.

Tech Tips*:*
1). Starting a new solution run with a restart file is always more time-efficient than starting an initial run. Thus, this "quick" method of obtaining a valid restart file can significantly shorten the time you will need to obtain a solution for the first run of a CFD simulation.

2). If you use different levels of sequencing to obtain restart files, be sure to create and save a new pgrx grid file from the original plot3d grid file to match the number of points in the restart file used for each DPLR run. See Section 7.1 and 7.2 for more information on DPLR Workflow and Workflow Shortcuts.

Chapter 4 – Using DPLR

Contents

4.0 Introduction

DPLR2D and DPLR3D are the main CFD solver applications provided in the DPLR Code Distribution Package. The two programs are closely related - sharing a common input deck format and most of the physics and numeric subroutines and libraries. However, two-dimensional or axisymmetric problems must be solved (and run much faster) with the DPLR2D executable whereas DPLR3D must be used for solving three-dimensional problems.

For this manual, the term DPLR will be used to refer to whichever solver application (DPLR2D or DPLR3D) is chosen for the problem under consideration.

4.1 Running DPLR

Step 1: **Open** the text editor program for your system.

Action: At the command line prompt, open:

/[path to your cfdinput directory]/ `generic.inp`

Result: A generic input file or "deck" appears on screen, with place-holder default values. To start with a blank deck, remove the values as shown on the following page.

Step 2: **Enter** appropriate, problem-specific values for each of the input variables or "flags". (See Section 4.3 for a description of DPLR input flags and a list of allowable values.)

Action: For each flag, type:

```
allowable, problem-specific value
```

Result: Input deck contains sufficient information for DPLR to process an input grid file and develop a solution to the problem.

Tech Tip: *Take special care to preserve the line spacing in the file as you enter new values and/or replace default values with problem-specific ones. If lines are added to or subtracted from the input deck file, DPLR will not be able to read it accurately.*

```
INPUT DECK FOR DPLR2D/DPLR3D CODE v4-01-1

gname,fname,bname,rname,dname,cname
'mygridname'
'myrestartname'
'mybcname'
'myradname'
‘myconnectname’
'PATH/cfdinput/air5sp5.chem'

     nblk     igrid     irest      ibcf     iradf     nfree     iinit

     ivis       ikt       ikv     ivmod     idmod     itmod     islip  iblow

   icatmd    ireqmd     twall      epsr    gamcat     xxxx     vwall

    ichem      ikeq      ivib      irot      ieex       iel      irad   ipen

   itrmod    itrans     trloc     trext     itshk

    istop     nplot     iplot      iaxi      ires

    igdum       kbl       kdg    istate     iresv

   xscale       ils     Le/Sc   LeT/ScT      prtl     prtlT

   xxxx         xxxx     rvr    resmin

================================================================
 SPACE MARCHING 1D IMPLEMENTATION
================================================================

   ispace     dxmin   slength     nxtot


================================================================
 TIME ACCURATE & STATISTICAL OPTIONS
================================================================

itime     lmax     dttol    tfinal   tfac

ifstat    iaero


================================================================
 GRID ADJUSTMENT/ALIGNMENT/MORPHING
================================================================

  igalign    ngiter    nalign     i1stadpt

  imedge    imradial   ngeom     ismooth
```

```
  fs_scale   ds_mult   gmargin

    ds1    cellRe     ds1mx     ds2fr

================================================================
 OVERSET GRID IMPLEMENTATION
================================================================

   iover     ioint    xxxxx

================================================================
 BLOCK #1
================================================================

   ntx     nty      ntz    iconr     isim    ifree   initi  ibadpt

   iflx      iord      omgi      ilim     idiss      epsi

   jflx      jord      omgj      jlim     jdiss      epsj

   kflx      kord      omgk      klim     kdiss      epsk

   iextst    nrlx    ildir     ibcu    iblag    ilt  ibdir   cflm

  Boundary condition type [ibc]:
    imin imax jmin jmax kmin kmax

================================================================
 Freestream Specification #1
================================================================

     irm    density    M/Re/V     cx      cy      cz

     Tin      Trin       Tvin      Tein

     turbi     tkref

     subp0     subT0      pback

      cs        (Species order: N2 O2 NO N O)

================================================================
 List of CFL numbers or timesteps for ramping
================================================================

-1
```

Figure 4-1 DPLR Input Deck

Step 3: **Save** the input deck file.

Action: At the command line, type:

```
save 'yourdplrinputfilename.inp'
```

Result: The input deck for your problem is saved.

Step 4: **Run** DPLR.

Action: At the command line prompt, type:

```
mpirun – np X (-machinefile machine.inp)
$path/dplr2d (or dplr3d) <
yourdplrinputfilename.inp
```

Result: DPLR performs the simulation on 'X' number of processors for the number of iterations specified in the input deck to achieve a solution. During the run, diagnostic output called the "standard out" (STDOUT), is echoed to the screen to provide feedback on the action(s) being performed, including any warning messages. If a fatal error is encountered, a descriptive message will be displayed in the STDOUT and the run will terminate.

When a specified convergence level is reached or you halt the run, an output solution (or restart) file `[.pslx]` is created along with an on-screen run summary. (See Section 4.3 for an example of a problem-specific DPLR input deck and run summary.)

Tech Tips:

1) *The run command above works for MPICH with a single type of processor. Different commands may be required for different MPI implementations or execution on heterogeneous clusters. Consult your system administrator for details on MPI program execution on your particular machine.*

2) *If a machinefile is required by your computer architecture, it consists simply of an ASCII listing of available machine (node) names, followed by the number of processes to start on each node. Example:*

node001:2	*or*	*node001 slots=2*
node002:2		*node002 slots=2*
node003:2		*node003 slots=2*

Both machine files show that the system can accept a job requiring up to 6 processors.

3) *To avoid slow performance, hangs or crashes, be sure that:*
- * *Nodes listed in the machine file are available for use and free of other jobs*
- * *Your job does not require more processors or memory than the machine file says the system can accept.*

4.2 Input Flags for DPLR

Input flags for DPLR are discussed below in the order they appear in the deck.

Input Filenames - These are external input files used by DPLR at runtime. Although these files can be specified using relative or absolute pathnames (with the exception of the chemistry input file which requires an absolute pathname), you may find that placing them in your problem-specific working directory creates a more productive computational environment for DPLR solutions.

Depending on the format of the file, a standard suffix will be assumed. See Section 9.2 for a list of file types and associated default suffixes.

`gname` – Specifies the name of the input XDR parallel grid file, and will typically have the suffix "`.pgrx`". If the file was prepared using FCONVERT, the name is specified in the `oname` flag of the FCONVERT input deck. This file contains not only the *xyz* coordinates of all the grid blocks in the simulation, but also information about block connectivity and the desired decomposition for processing. This file is required and must already exist when the simulation run begins.

`fname` – Specifies the name of the input restart file, and will typically have the suffix "`.pslx`". This is a required file name. If this is a new simulation, DPLR will create the file to go with the name specified here. If the simulation is a rerun, the file specified here should already exist. See Section 6.6 for more information on restart files.

`bname` – Specifies the name of the input boundary condition file, and will typically have the suffix ".`pbca`". This file is not

required to run a simulation, but having one gives you increased flexibility in specifying point-by-point parameters as opposed to the standard block face parameters. See Section 6.4 for more information on boundary condition files.

rname – Specifies the name of the input surface radiation file, and will typically have the suffix ".prdx". This file is optional and read only if volumetric radiation data are input and the irad flag is set to =1. If the file is not required for the simulation, use "none" as the filename.

dname – Specifies the name of the input overset connectivity file and will typically have the suffix ".dci" if ioint=1. This file is only required in iover=1. If overset logic is not enabled, use "none" as the filename.

cname – Specifies the name of the input chemistry file, and will typically have the suffix ".chem". This file is required and must exist in the "cfdinput" directory that is created when you install the DPLR Code Package. See Section 2.4 for information on the directory and file structure of the DPLR Code Package and Section 6.7 for more information on chemistry files.

Tech Tip: *Unlike other input files, the absolute pathname to this file must be specified in the input deck.*

Global Modeling Flags – These flags are for values that remain constant for all blocks of the simulation.

nblk – Specifies the number of master grid blocks in the simulation. This is the same value as nborig in the FCONVERT input deck and will be less than or equal to the number of processors on which the job is run.

igrid – Specifies the format of the input grid file (gname). Allowable values are:

1 Parallel archival file (native unformatted)

11 Parallel archival file (XDR format) *(Recommended)*

21 Parallel archival file (ASCII)

irest – Specifies the format of the restart file (`fname`). Allowable values are:

1 Parallel archival file (native unformatted)

11 Parallel archival file (XDR format) *(Recommended)*

21 Parallel archival file (ASCII)

ibcf – Specifies the format of the boundary condition (BC) file (`bname`), if any. Allowable values are:

0 Do not read a BC file

1 Parallel archival file (native unformatted)

11 Parallel archival file (XDR format)

21 Parallel archival file (ASCII)

iradf – Specifies the format of the input radiation file (`rname`), if any. Allowable values are:

0 Do not read a radiation file

1 Parallel archival file (native unformatted)

11 Parallel archival file (XDR format)

21 Parallel archival file (ASCII)

nfree – Indicates the number of freestream specifications (i.e. areas of the freestream with preconfigured flow conditions) that are characterized in the DPLR input deck.

iinit – Specifies how to initialize the simulation. Allowable values are:

0 Start all blocks by initializing to freestream values in the specification identified in the block

1 Restart from saved file

2 Start with a stagnant interior at low pressure

3 Start with artificial boundary layer in place

10 Block-by-block initialization using `iconr` flag

11 Restart from saved file, reset `nit` and `etime`

Tech Tip: *When you are running a simulation for the very first time, set iinit=0. Thereafter, set iinit = 1 unless problem specifics require use of one of the other available options.*

ivis – Specifies the equation set to solve. Allowable values are:

0	Euler simulation (neglect Navier-Stokes terms)
1	Laminar full Navier-Stokes simulation
2	Turbulent full Navier-Stokes simulation
11	Laminar Navier-Stokes simulation (thin-layer)
12	Turbulent Navier-Stokes simulation (thin-layer)

Tech Tips:
1) *DPLR is a full Navier-Stokes solver and recommended to be used as such. By setting* `ivis=0`*, you can run the problem in Euler mode, but the run time per iteration will increase significantly and viscous boundary conditions should not be specified.*
2) *Running DPLR in thin-layer mode is also not recommended because there are no time or memory savings in doing so.*
3) *The turbulence model to be employed when* `ivis=2` *is determined by the* `itmod` *flag.*

ikt – Specifies the model used to compute translational thermal conductivity. An appropriate setting is required for all viscous simulations. Allowable values are:

1	Use the model that is consistent with `ivmod`
2	Use constant Prandtl number expression

Tech Tip: `ikt=1` *is the preferred setting for all practical applications.*

ikv – Specifies the model used to compute vibrational thermal conductivity. An appropriate setting is required for all viscous simulations with vibrational nonequilibrium (`ivib`=1, 3, 4). Allowable values are:

1	Standard expression with e_v gradients
2	Hard sphere approximation with e_v gradients

11 Standard expression with T_v gradients *(preferred setting for all practical applications.)*

12 Hard sphere approximation with T_v gradients

Tech Tips:
1). Hard sphere approximations are provided only for comparison to legacy codes and should not be used.
2). The choice between e_v and T_v gradients is somewhat arbitrary, and scales the resulting vibrational thermal conductivity (k_v) by the vibrational specific heat ($C_{v_{vib}}$):

$$q_v = \kappa \frac{\partial T_v}{\partial \eta} \approx \kappa \frac{\partial T_v}{\partial e_v} \frac{\partial e_v}{\partial \eta} = \kappa' \frac{\partial e_v}{\partial \eta}$$

$$\kappa'_v = \kappa / C_{v_{vib}}$$

For most simulations there is little difference between ikv = 1 and ikv = 11. However, for cases where the flow is nearly completely dissociated, using energy gradients becomes slightly unstable because there is little energy in this mode.

`ivmod` – Specifies the baseline model used to compute mixture viscosity and thermal conductivity. An appropriate setting for `ivmod` is required for all viscous simulations. Allowable values are:

1 Blottner/Wilke model with an Eucken relation *(inaccurate at elevated temperatures)*

2 Sutherlands Law and constant Prandtl number *(available only for perfect gas flows but a reasonable estimate at low to moderate temperatures)*

3 Yos approximate mixing rules *(preferred model for all reacting gas simulations)*

4 Full first-order Chapman Enskog multicomponent *(NOT WORKING in DPLR 4.01.1)*

11 Blottner/Armaly-Sutton with an Eucken relation *(requires composition-dependent tailoring of the free parameters for maximum accuracy)*

12 Keyes' Equation and constant Prandtl number *(should be used only where the freestream temperature is very low (<100K)*

Tech Tip: `ivmod=3` *has been shown to be a reasonable and general approximation to the true Chapman-Enskog fluxes.*

`idmod` – Specifies the baseline model used to compute species diffusion coefficients. An appropriate setting for `idmod` is required for all multi-species viscous simulations. Allowable values are:

0 No diffusion *(single species)*

1 Constant Lewis/Schmidt number *(assumes all species have the same diffusion coefficient)*

2 Bifurcation model *(developed to model boundary layer diffusion of carbon-based ablators; requires as input least squares fit coefficients for each species)*

3 Self-Consistent Effective Binary Diffusion *(preferred model for all multi-species calculations, but somewhat unstable for separated flows. See Tech Tip below.)*

5 Iterative multicomponent *(NOT WORKING in DPLR 4.01.0)*

11 Constant Lewis/Schmidt number, ignore ambipolar diffusion *(widely used, but often inaccurate; provided for comparison to heritage codes.)*

12 Bifurcation model, fits obtained with assumption of ambipolar diffusion *(developed to model boundary layer diffusion of carbon-based ablators; appropriate when using coefficients provided by Olynick)*

13 Self Consistent Effective Binary Diffusion, ignore ambipolar diffusion *(provided for comparison to heritage codes)*

Tech Tip: *The preferred model for all multi-species calculations is the Self-Consistent Effective Binary Diffusion (SCEBD) model of Ramshaw and Chang (*`idmod`*=3,13), which has been shown*

to give results in good agreement with exact solutions of the Stefan-Maxwell equations. This model requires as input collision integral data for each binary interaction in the mixture. These data are imported to DPLR via the "`gupta.tran`" physical model file in the `cfdinput` directory. However, it does tend to be unstable for separated flows, particularly while the recirculation region is being formed. Therefore, *for separated flows, start the solution with `idmod=1` and an appropriate Schmidt number, then switch to `idmod=3` once the flow structures have stabilized.*

`itmod` – Specifies the turbulence model to be employed. An appropriate setting for `itmod` is required for all turbulent simulations indicated by `ivis=2,12`. Allowable values are:

0	Laminar Flow *(non-turbulent)*
1	Baldwin-Lomax Model *(reasonable results for attached flows with a favorable pressure gradient on both blunt and slender bodies)*
1000	Spalart-Allmaras Model *(no compressibility correction)*
1001	Spalart-Allmaras Model *(Catris&Aupoix comp.)*
1002	Spalart-Allmaras Model *(Secundov comp.)*
1050	Detatched Eddy Simulation with Spalart-Allmaras Model *(Hybrid Reynolds Average Navier Stokes (RANS) Large Eddy Simulation (LES) model – no compressibility correction)*
1051	Detatched Eddy Simulation with Spalart-Allmaras Model *(Hybrid RANS LES model – Catris&Aupoix comp.)*
1052	Detatched Eddy Simulation with Spalart-Allmaras Model *(Hybrid RANS LES model – Secundov comp.)*
2001	Menter SST Model *(no compressibility correction)*
2002	Menter SST Model *(compressibility correction #1)*

2003	Menter SST Model *(compressibility correction #2 - recommended model for separated flows and flows with adverse pressure gradients)*
2004	Menter SST Model *(compressibility corrections with further modifications)*
2011	Overflow 2.0a "version" of SST
2021	Menter SST 2006 K-W model *(no compressibility)*
2022	Menter SST 2006 K-W model *(compressibility correction #1)*
2023	Menter SST 2006 K-W model *(compressibility correction #2)*
2024	Menter SST 2006 K-W model with further modifications and improvements
2051	Detatched Eddy Simulation with Menter SST Model *(Hybrid RANS LES model – no compressibility correction)*
2052	Detatched Eddy Simulation with Menter SST Model *(Hybrid RANS LES model –compressibility correction #1)*
2053	Detatched Eddy Simulation with Menter SST Model *(Hybrid RANS LES model - compressibility correction #2 - recommended model for separated flows and flows with adverse pressure gradients)*
2054	Detatched Eddy Simulation with Menter SST Model with further modifications and improvements
3001	1998 K-W modified 3-equation, for Lag development *(NOT WORKING in DPLR 4.01.1.)*
3002	Lag Turbulence model *(NOT WORKING in DPLR 4.01.1)*

`islip` – Specifies the model to be used for slip-wall boundary conditions. Allowable values are:

0	Disable wall slip *(recommended setting. See Tech Tip below.)*
1	Maxwellian slip model

Tech Tip: *Slip walls are not generally employed for simulations in the hypersonic or supersonic continuum. DPLR has currently implemented velocity and temperature slip models, but not species density (mole fraction) slip conditions. The slip wall model currently used in DPLR has not been fully validated yet, and should therefore be used with caution.*

`iblow` – Specifies the model to be used for blowing-wall boundary conditions. Allowable values are:

0 Disable wall blowing *(usual setting)*

1 Specified wall blowing velocity (m/s)

2 Specified unit mass flow rate ($kg/m^2/s$)

Tech Tip*: If blowing wall boundary conditions <u>are</u> taken into account,* **`vwall>0`** *defines a blowing wall and* **`vwall<0`** *defines a sucking wall. Pointwise blowing rates can be specified using the pointwise boundary condition file (.pbca) (See Section 6.3 for more information on boundary condition files). Complex blowing models can be characterized using the material response boundary conditions* **`icatmd`**, **`ireqmd`**, **`twall`**, **`epsr`**, *and* **`gamcat`** *discussed below,*

`icatmd` – Specifies the model to be used for wall catalysis. Allowable values are:

0 Disable wall catalysis *(wall is assumed to be non-catalytic and the gradient of all species mole fractions is assumed to be zero.)*

1 Constant γ, homogeneous model *(value for γ must be with the* **`gamcat`** *flag.)*

2 Constant γ, fully catalytic to ions but supports only homogeneous surface reactions such as $N + N \rightarrow N_2$ & $O + O \rightarrow O_2$.

3-98 Material specific surface kinetics *(reaction rates for different materials are experimentally obtained and given in the "***`catalysis.surf`***" file in the* **`cfdinput`** *directory)*

99	Input material map *(allows DPLR to access a surface catalytic map specifying pointwise material properties in a boundary condition file)*
100	Supercatalytic wall *(assumes chemical composition at the wall is identical to the freestream, resulting in conservative enthalpy estimates appropriate for design studies)*
101-198	Supcercatalytic with specified freestream *(most appropriate for high enthalpy simulations in ground test facilities)*
200	Mitcheltree CO_2 model *(developed for the Mars-like $C0_2$ atmosphere; models diffusion limited recombination pathways)*
201	Enhanced Mitcheltree CO_2 model *(additional recombination pathways in this model may make it more appropriate for higher flow enthalpies.)*
300	Generalized CO_2 catalysis (Bose/Wright) *(used in sensitivity analyses, not design work)*
999	Equilibrium chemistry boundary condition
1001	Parks Cabron Oxidation/Sublimation
1002	Shuttle STS Carbon Oxidation

Tech Tip: *Catalysis refers to how the wall surface facilitates chemical reactions that can deposit energy on the vehicle surface during flight. Much work is currently being done to enhance empirical knowledge of material responses under hypersonic flight conditions in a variety of atmospheric flows. As the materials knowledge base increases, DPLR can be used to simulate how the overall flow environment contributes to and, in turn, is affected by chemical reactions on flight vehicle surfaces.*

`ireqmd` – Specifies the model to be used for surface radiative equilibrium. Allowable values are:

0	Disable radiative equilibrium
1	Constant emissivity (ε) model *(value for ε set with the `espr` flag)*

3-98	Material specific ε *(emission rates for different materials are experimentally obtained and given in the "*`emission.rad`*" file in the* `cfdinput` *directory)*
99	Input material map *(allows DPLR to access an emission radiation map specifying pointwise material properties in a boundary condition file)*
101-199	Material specific ε with a maximum wall temperature *(used to model physical material temperature limits during initial design analysis. Maximum temperature specified with* `twall` *flag. DPLR will automatically switch between an isothermal and radiative equilibrium wall on a pointwise basis if this option is used.)*
201-209	Material specific ε with a view factor correction *(allows you to modify the surface emissivity in a pointwise manner to enable a simple view-factor correction to the hemispherical emissivities for internal and/or cavity flows.)*

Tech Tip: *A radiative equilibrium wall is a common design model that assumes all energy incident to the surface of a vehicle is reradiated to space at a rate consistent with the emissivity of the wall material. As the materials knowledge base increases, DPLR can be used to simulate how the overall flow environment contributes to and, in turn, is affected by radiative emissivity of vehicle surfaces during hypersonic flight.*

`twall` – Specifies the wall temperature to be used for isothermal temperature-capped radiative equilibrium wall simulations. Also specifies wall temperature if `ireqmd >100.`

`epsr` – Specifies the constant value of emissivity (ε) to be used for radiative equilibrium wall simulations (`ireqmd=1`)

`gamcat` – Specifies the constant value of catalytic effcienty (γ) to be used for catalytic wall simulations (`icatmd=1,2`)

xxxx – Not used in DPLR 4.01.0.

vwall – Specifies the wall velocity or blowing rate. Allows you to impose a constant blowing (`vwall>0`) or sucking (`vwall<0`) when used with the `iblow` flag. If `iblow=1`, expressed as velocity (m/s) . If `iblow=2`, expressed in mass flux per unit area (kg/m2/s).

ichem – Specifies the model employed for chemical reactions in the gas phase. Allowable values are:

0 Frozen chemistry *(no chemical reactions occur in the flowfield)*

1 Finite-rate chemistry *(Arrhenius style reaction kinetics used to model the chemistry are given in the "`.chem`" file specified with the `dname` flag and found in the `cfdinput/directory`.)*

Tech Tip: *DPLR does not currently support equilibrium chemistry.*

ikeq – Specifies the model used for computing equilibrium constants. This is required when `ichem=1.` Allowable values are:

-1 No reverse reactions *(debugging tool that turns off reverse reactions completely; should not be used for actual simulations)*

1 Park 1985 fits *(not recommended for use due to potential instabilities in some simulations)*

2 Mitcheltree 1994 fits *(not recommended for use due to potential instabilities in some simulations)*

3 Park 1990 fits ($n = 10^{16}/\text{cm}^3$) *(not recommended for use due to potential instabilities in some simulations)*

4 Park 1990 fits ($n = 10^{19}/\text{cm}^3$) *(not recommended for use due to potential instabilities in some simulations)*

9 Computed from NASA Lewis thermodynamic fits (1994) *(preferred method of computing*

equilibrium constants using Gibb's free energy method where species enthalpy and entropy are computed using curve fit expressions given by Gordon and McBride with the final equilibrium constant determined via the van Hoff't equation.)

11-19 Same as 1-9 with ramped limiter *(slowly approaches more aggressive limiter value ; for advanced users only)*

21-29 Same as 1-9 with aggressive limiter *(begins with larger limiter as default as used in simulations of the Fire-II entry vehicle at early trajectory points: for advanced users only)*

31-39 Same as 1-9 with conservative limiter *(scales down the 1-9 model selected by 75%)*

Tech Tip: *Equilibrium constants used to compute the reaction kinetics for chemical reactions in the gas phase are computed by default in DPLR via Arrhenius expressions. The curve fit model offered in* `ikeq=9` *has been found to yield the most stable equilibrium constants for most simulations. However, in those situations where the equilibrium constants are either very small or very large, e.g. ionized wake flow simulations and low density ionized flow simulations with non- or weakly catalytic cold walls, DPLR offers ways to minimize the potential solution instability that may occur with these extreme values. By setting* `ikeq=11-19`*, DPLR slowly increases the value of the constant limiter as the solution progresses. By setting* `ikeq=21-29`*, DPLR begins with a larger limiter as the default value to avoid computed heat transfer at the vehicle surface being too small. By setting* `ikeq=31-39`*, DPLR scales down the 1 – 9 model selected by 75%. Although extreme situations such as these are unlikely to occur during routine use of DPLR, the capability of dealing with them does exist for the advanced user who needs to do so.*

`ivib` – Specifies the model used to compute the vibrational energy component of the gas. Allowable values are:

0 Neglect vibrational energy

1 Vibrational nonequilibrium, single T_v *(recommended for all planetary and high-velocity Earth entry flows)*

2 Vibrational equilibrium using statistical mechanics

3 Complete thermal equilibrium using NASA LeRC curve fits *(based on data from NASA's Lewis Research Center's (now Glen Research Center) computer program CEA (NASA Reference Publication 1311). Can be employed for low altitude hypersonic flight or some Shuttle-type reentry trajectories where vibrational nonequilibrium is not very important)*

4 Two temperature model using LeRC curve fits ($T_r = T_{el} = T$) *(not recommended)*

5 Two temperature model using LeRC curve fits ($T_r = T$; $T_v = T_e = T_{el}$) *(NOT WORKING in DPLR 4.01.0)*

11 Virbrational nonequilibrium, multiple T_v *(NOT WORKING in DPLR 4.01.1)*

Tech Tip: *If* `ivib=3` *or* `4` *, the values of* `irot`*,* `ieex`*, and* `iel` *are ignored because these settings uniquely determine the apportionment of internal energies among the various modes.*

`irot` **-** Specifies the model used to compute the rotational energy component of the gas. Allowable values are:

1 Rotational nonequilibrium, single T_v *(mainly used for radiation studies of high altitude flows)*

2 Rotational equilibrium using statistical mechanics *(recommended setting)*

3 Complete thermal equilibrium using NASA LeRC curve fits *(based on data from NASA's Lewis Research Center's (now Glen Research Center) computer program CEA (NASA Reference Publication 1311).*

4 Two temperature model using LeRc curve fits ($T_r = T_{el} = T$; $T_v = T_e$) *(not recommended)*

5 Two temperature model using LeRC curve fits ($T_r = T$; $T_v = T_e = T_{el}$) *(NOT WORKING in DPLR 4.01.0)*

11 Rotational nonequilibrium, multiple T_r *(NOT WORKING in DPLR 4.01.1)*

***Tech Tip**: Unlike vibration, the rotational mode of the gas is assumed to be fully excited, and thus cannot be neglected for polyatomic species. You must decide whether to model the rotational mode in equilibrium with the translational mode (`irot = 2-4`) or in nonequilibrium (`irot=1`). In practice, it is rarely necessary to solve for a nonequilibrium rotational energy so this feature is provided mainly for detailed radiation studies of high altitude flows.*

`ieex` – Specifies the model used to compute the electronic energy component of the gas. Allowable values are:

0 Neglect electronic energy

1 Statistical mechanics ($T_e = T$) *(recommended setting)*

2 Statistical mechanics ($T_e = T_v$)

3 Complete thermal equilibrium using NASA Lewis curve fits *(based on data from NASA's Lewis Research Center's (now Glen Research Center) computer program CEA (NASA Reference Publication 1311).*

4 Two temperature model using LeRC curve fits ($T_r = T_{el} = T$; $T_v = T_e$) *(not recommended)*

5 Two temperature model using LeRC curve fits ($T_r = T$; $T_v = T_e = T_{el}$) *(NOT WORKING in DPLR 4.01.1)*

***Tech Tip**: For `ieex=1`, the contribution of the electronic energy to the total is computed using statistical mechanics based on characteristic temperatures and degeneracies in the "`chemprops.spec`" file from the `cfdinput` directory and is assumed to be in equilibrium with the translational mode.*

iel – Specifies the model used to compute the free electron energy component of the gas. Allowable values are:

- 0 Neglect free electron energy (*only valid for flows with no ionization*)
- 1 Coupled free electron and translational energy ($T_{el} = T$) *(recommended setting ; assumes that the energy of the free electron gas is governed by the translational temperature)*
- 2 Coupled free electron energy and vibrational energy ($T_{el} = T_v$) *(NOT WORKING in DPLR 4.01.1)*
- 3 Complete thermal equilibrium using NASA LeRC curve fits *(based on data from NASA's Lewis Research Center's (now Glen Research Center) computer program CEA (NASA Reference Publication 1311).*
- 4 Two temperature model using LeRc curve fits ($T_r = T_{el} = T$; $T_v = T_e$)
- 5 Two temperature model using LeRc curve fits ($T_v = T_e = T_{el}$; $T_r = T$) *(NOT WORKING in DPLR 4.01.1)*
- 11 T_{el} Independent *(NOT WORKING in DPLR 4.01.1)*

irad – Specifies the model used for shock layer radiation modeling. Allowable values are:

- 0 No radiation model *(for weakly radiating flow fields)*
- 1 Read pointwise $\Delta \cdot Q_R$ from a file *(use only if* `rname` *is defined.)*
- 2 Optically thin emission (Carbon-Nitrogen Violet)
- 3 Optically thin emission (Carbon-Nitrogen Red)
- 4 Optically thin emission (Carbon-Nitrogen Violet + Red)
- 102-198 Same as 2-98 but with input surface heating information read from the `rname` file

Tech Tip:
1). *DPLR does not compute shock layer radiation directly, but several hooks are provided for coupling, either loosely to external radiation transport codes or tightly for optically thin emission. Typically, for weakly radiating flowfields, shock layer radiation is either neglected or computed in an uncoupled manner. For these cases,* `irad` *= 0.*

2)*. If the radiation field is known to be optically thin, DPLR supports tight coupling by computing the* $\Delta \cdot Q_R$ *source term at each volume cell using curve fits generated by comparison to more exact computations. Currently DPLR supports this option for CN radiation only (irad = 2-4). In this case, DPLR reads the curve fit coefficients from the file "*`emission.rad`*" from the* `cfdinput` *directory. As information becomes available for other species, this file can be updated and expanded.(Note: this option assumes that energy converted to radiation is instantly lost from the control volume*.)

3). *To include the surface radiative heating effects in the radiative equilibrium surface energy balance, set* `irad=102-104` *and read the pointwise surface radiative heating from a radiation file (*`rname`*) in your working directory. See Section 6.7 for more information on radiation files.*

4). *To use an external radiation transport code such as RADEQUIL or NEQAIR to loosely couple flowfield radiation information to DPLR solutions, iterate between the two codes as follows:*

- ***a****. Set irad=0 and run an initial DPLR solution.*
- ***b****. Extract data using POSTFLOW for input to your radiation transport code.*
- ***c****. Create a radiation file (*`rname`*) from data generated by running your radiation transport code. See Section 6.7 for more information on radiation files.*
- ***d****. Perform a second DPLR run with irad=1.*
- ***e****. Repeat until you obtain a fully converged solution.*
- ***f****. Compare your first and final solutions to determine the significance of the shock layer radiation value.*

`ipen` – Specifies the model used for reaction product energy distribution. Allowable values are:

0	All species are created/destroyed at the internal energy of the mixture
1-9	n*10 percent of dissociation energy for all reactions *(NOT WORKING in DPLR 4.01.1)*

`itrmod` – Specifies the model used for turbulence transition modeling. This flag is used whenever a turbulent flowfield is specified by `ivis=2, 12`. Allowable values are:

0	Neglect transition, flow is fully turbulent
1	TANH transition function
2	Dhawan and Narashima model
3	Sigmoid function
100	Input transition map *(allows simulation of local turbulent regions in a laminar flow)*

Tech Tip: *Although some turbulence models in DPLR are capable of predicting transition, the code also allows you to use several models to impose transition in locations of your choosing. When* `itrmod=1,2,3` *, the location and extent of the transition regions are defined by the values you put into* `itrans`*,* `trloc`*, and* `trext`*. When* `itrmod=100`*, you have the option of creating a transition map consisting of a turbulence intensity value ranging from 0 (fully laminar) to 1 (fully turbulent) at each surface point and placing the information into a boundary condition file (*`bname`*). See Section 6.3 for more information on boundary condition files. Remember, however, choosing <u>any</u> of these models will impose, and not predict, turbulent transition(s) in your simulation.*

`itrans` – Specifies the ordinate of the transition onset location. This flag is used whenever a turbulent flowfield (`ivis=2,12`) with an input transition model (`itrmod=1,2,3`) is chosen. Allowable values are:

±1	Transition at specified constant *x* value
±2	Transition at specified constant *y* value
±3	Transition at specified constant *z* value

Tech Tip: *Setting* `itrans` *to a positive value implies transition proceeds with increasing ordinate, while setting it to a negative*

value implies that transition proceeds with a decreasing ordinate.

trloc – Specifies the transition onset location. This flag is used whenever a turbulent flowfield (`ivis=2,12`) with an input transition model (`itrmod=1,2,3`) is chosen. `trloc` is a real dimensional number tied to the value of `itrans`. For example, if `itrans=1` and `trloc=2.5`, DPLR will initiate transition at a value of x=2.5m, with turbulent flow for larger values of x and laminar flow for smaller values.

trext – Specifies the extent of transition to turbulent flow. This flag is used whenever a turbulent flowfield (`ivis=2,12`) with an input transition model (`itrmod=1,2,3`) is chosen. `trext` is a real dimensional number equal to the width of the TANH function from 0.01 – 0.99. The other transition models do not permit user modification of the transition length, so the `trext` flag is not used in those cases.

itshk – Specifies the type of limiting employed (Spalart-Allmaras & SST) for turbulent shock interaction. Allowable values are:

0 No shock interaction modification

1 Standard limiting, Wilcox-type for SST or SALSA for S-A

2 Limiting with low Reynolds number effect, for SST only

istop – Specifies the number of iterations to run before stopping the simulation. This is a relative value. For example, if a simulation is started after 500 iterations are already complete and `istop=100`, DPLR will run 100 additional iterations, reaching completion after 600 total iterations.

Tech Tip: *DPLR can also be instructed to stop a run when a specified L2norm residual level is reached using the* `resmin` *flag. In this case, termination will occur when the first condition, either* `istop` *or* `resmin`*, is met.*

nplot – Specifies the frequency of restart file writes. DPLR will save a restart file periodically every `nplot` iterations during the solution run as long as the flag `iplot>0`.

Tech Tip: *A good general value to use for* `nplot` *is usually 100 - large enough so DPLR does not spend a large percentage of the runtime writing restart files, but small enough so a lot of work is not lost if the job quits for some reason.*

iplot – Controls the redundancy of restart file writes. Allowable values are:

-1 or 0	Do not write a restart file *(use only for debugging)*
1	Write a single restart file
n	Save *n*-1 prior restarts
-99	Force restart file write *(use only for debugging)*

Tech Tip: *Setting* `iplot` *to a positive integer larger than 1 (n) causes DPLR to save n-1 previous restart files in addition to the current one. To distinguish saved files, DPLR will append the iteration number of the restart file to the filename specified in* `fname`*. For example,* `if fname=sample.pslx, iplot=3, nplot=100`*, then after 1000 iterations the following files will exist: sample.pslx, sample.pslx-900, sample.pslx-800. Note that older restart files, created every 100 iterations as specified by the value in* `nplot,` *are not saved!*

iaxi – Enables axisymmetry in a DPLR2D simulation run. Allowable values are:

-1 or 0	Non-axisymmetric (2D)
1	Axisymmetric about the *x*-axis
2	Axisymmetric about the *y*-axis

Tech Tip: *DPLR2D simulates axisymmetric flows by solving the Navier-Stokes equations in cylindrical rather than Cartesian coordinates. This allows for an axisymmetric simulation in about the same total solution time as a 2D result. The rotation axis of the problem is always assumed to be either the x- or y-axis. Note*

that DPLR2D simulations are always in the xy-plane, so rotation about the z-axis is not permitted.

ires – Specifies the type of residual and convergence data that are tracked and output to the screen and to the convergence file. Allowable values are:

0	Do not output a convergence file
1	Output nit, global residual, and Δt *(iteration number, summed residual over all computational blocks, timestep for the iteration number)*
2	Output nit, global residual, and CFL number
3	Output nit, global residual, and CPU time
4	Output nit, global residual, and flow time
5	Output nit, global residual, CFL number, and aero data
11	Output nit, block residual, and Δt
12	Output nit, block residual, and CFL number
13	Output nit, block residual, and CPU time
14	Output nit, block residual, and flow time
15	Output nit, block residual, CFL number, and aero data
22	Output nit, global residual, and min/max CFL
32	Output nit, block residual, and min/max CFL
99	Output nit, global residual, CFL number, and Δt

Tech Tips:
1)) `nit` *= iteration number;* `global residual` *= summed residual over all computational blocks;* Δt *= timestep for the iteration number;* `CFL number` *= CFL number for the iteration number;* `CPU time` *= elapsed CPU time at the iteration number;* `flow time` *= elapsed flow time at the iteration number (only useful for time accurate simulations);* `aero data` *= data written to a* `*.aero` *file for each iteration when viscous fluxes are included in the simulation;* `block residual` *= residuals computed for each master block in the simulation, (written only to the convergence file).*

2) *To compare the output residual at each iteration with the computed residual in the first iteration, enter the* `ires` *value as a negative number. Although this technique is usually preferred by DPLR users for viewing progress toward solution conversion, you should be aware that certain problems can have very small or zero residuals in the first iteration which would result in seemingly large, or inappropriate, residuals at later iterations.*

3) *Although DPLR will automatically capture specific information in the convergence file ".con" and in the log file ".log", these are both subsets of what is echoed to the screen during a DPLR run. To capture all the information created during a DPLR run, you can set up your own user log file to run in the background by typing the following at the command prompt:*

```
mpirun – np X (–machinefile machine.inp)
/$path/dplr2d (or dplr3d)<
yourdplrinputfilename > userlogfile &
```

This will result in your system returning you to the command prompt and capturing the screen data in the background for your future reference. See Section 6.8, 6.9, and 6.10 for more information on convergence, aerodynamic, and log files.

`igdum` – Controls the computation of grid dummy cell coordinates. Allowable values are:

0	Only compute if necessary *(preferred setting)*
1	Always recompute *(use if boundary conditions are changed during simulation e.g. if a setup error is detected)*
-99	Only compute if necessary, output debugging files *(for use by code developers)*
99	Recompute and output debugging files *(for use by code developers)*

Tech Tip: *When a grid file is created by FCONVERT, dummy cell values are not computed because FCONVERT does not have information about the correct boundary conditions to enforce at each grid face. DPLR will automatically generate the correct dummy cell coordinates for each block based on the supplied*

boundary conditions, and send the correct data to all processors in the simulation. DPLR will then overwrite the stored grid file to include the dummy cell information. The preferred setting for this flag is `igdum=0` *because there is usually no need to recompute dummy cells unless an error is detected and the boundary conditions change during the simulation (*`igdum=1`*)*

`kbl` – Seldom needed, rarely used. (Allows you to zero out the body-normal added dissipation term in the boundary layer. If `kbl` is a positive integer, the body-normal eigenvalue will be zeroed out for the `kbl` cells nearest to each solid wall in the simulation and smoothly increased to the specified value.)

`kdg` – Tangential epsilon augmented near axis boundary condition for k<kbl. Must have one or more 1011:1019 or 1011:2019 BCs for this to have any effect. For developers only. Leave set to 0.

`istate` – Specifies equation of state being used. Allowable values are:

0, 1	Perfect gas
2	Real gas (excluded volume) *(NOT WORKING in DPLR 4.0)*

`iresv` – Controls the residual variable(s) tracked by DPLR. Allowable values are:

1	Total density *(sum of L2Norm of all species densities)*
2	Velocity *(sum of velocity components)*
3	Energy *(sum of energy equations)*
4	Turbulence variables *(sum of turbulence variables for Spalart-Allmaras or Menter SST model)*
-n	Conserved variable #n *(See Tech Tip below)*

Tech Tip: *You can track the residual of a single equation by using a negative integer for* `iresv`*. For example, for a 5-species 2D simulation, the residual in the u momentum component can be tracked with* `iresv` *= -6. Since DPLR is a*

fully coupled code (with the exception of some turbulence models), convergence of one variable is typically dependent on convergence of the others, which limits the utility of single variable residual. However this option can be useful for analyzing an unstable simulation because the offending equation will generally "blow up" before the others do.

`xscale` – Used to scale the input grid at runtime. Allowable values are:

1	No scaling *(recommended)*
f	Multiply grid dimensions by this value immediately after read

Tech Tip: *DPLR will print a warning message if* **`xscale`** *is not set to 1.*

`ils` – Specifies whether input constants governing laminar (Le/Sc) and turbulent (LeT/ScT) diffusion coefficients are to be interpreted as Lewis or Schmidt numbers. Allowable values are:

1	Lewis Number
2	Schmidt Number

`Le/Sc` – Specifies the value of the laminar Lewis or Schmidt number to be employed in the simulation. This parameter is relevant for viscous simulations (`ivis` $\neq$ `0`) with constant Lewis/Schmidt number diffusion (`idmod =1,11`). Choosing a constant Schmidt number is typically preferred, with appropriate values varying with the target destination and entry velocity. Recommended values:

0.4 – 0.7

Tech Tip: *Remember that the preferred approach is to model multi-species diffusion coefficients (`idmod=3`), in which case this flag is not used during the simulation.*

LeT/ScT - Specifies the value of the turbulent Lewis or Schmidt number to be employed in the simulation. Relevant for turbulent viscous simulations (`ivis=2,12`) regardless of the model used to compute species diffusion coefficients set by the `idmod` flag. Recommended values:

0.5– 1.0

Tech Tip: *A value of 0.7 has been baselined for the Mars Science Laboratory.*

prtl - Specifies the value of the Prandtl number to be employed in the simulation. Relevant for viscous simulations (`ivis`$\neq$`0`) with constant Prandtl number thermal conductivity model (`ivmod=2,12` or `ikt=2`.) Recommended value for low temperature air flows:

0.72

Tech Tip: *These models should only be selected for perfect gas (non-reacting) low temperature flows. As such the value of* `prtl` *is not usually relevant.*

prtlT - Specifies the value of the turbulent Prandtl number to be employed in the simulation. Relevant for all turbulent viscous simulations (`ivis=2,12`) irrespective of the turbulence or laminar conductivity model. Recommended value:

0.9

xxxx – Not used in DPLR 4.01.1

xxxx – Not used in DPLR 4.01.1.

rvr - Viscous overrelaxation parameter. Recommended value:

1.3 *(default)*

resmin – Specifies the minimum L2Norm residual for DPLR to reach to achieve a converged solution. Recommended value:

1×10^{-8} or lower

Tech Tip: *If you prefer to run your simulation to a specified number of iterations irrespective of the residual, set* `istop` *to the desired number of iterations and* `resmin` *to a very small value, such as* 1×10^{-20}.

Space Marching 1D Implementation – Each of the flags in this portion of the DPLR input deck allow for shock capture in 1D only.

ispace – Used for 1D space marching simulations, i.e., shock tube flows. Allowed values are:

0 Disable space marching
1 Enable space marching

Tech Tip: *Seldom used in practice, this option gives you a fast and efficient tool to perform 1D flow simulations with complex models. If this option is used, many of the other flags in the code are ignored.*

dxmin – Sets the minimum *x*-spacing for the space marching routine. *(Only used when* `ispace=1`*)*

slength – Sets the total marching distance for the space marching routine. *(Only used when* `ispace=1`*)*

nxtot – Sets the total number of cells for the space marching routine. *(Only used when* `ispace=1`*)*

Time Accurate and Statistical Options Flags – Prior to release version 4.01.1, DPLR has been primarily used to solve steady state problems characterized by large, but stable time steps. With this release, DPLR can be run in a time-accurate fashion and is thus capable of studying transient phenomena. Although default values are suggested for each of the input flags, users should determine context-appropriate values for each simulation problem employing this capability.

itime – Specifies time integration order of accuracy. Allowable values are:

0 1st order of accuracy

1 2nd order (dual time-stepping) accuracy

lmax – Specifies maximum number (*n*) of sub iterations per time step when `itime=1.` Default value:

5

dttol – Specifies the residual tolerance (f) for convergence of the sub iterations when `itime=1.` Default value:

10^{-3}

tfinal – Tells DPLR to stop the simulation when flow time reaches this (f) value. Default value:

10^{99} [very large number]

tfac – Specifies the multiplicative factor on physical time step used to determine dual time step (i.e. sub iteration) when `itime=1.` Default value:

10^{15} [very large number]

ifstat – Specifies flow statistics DPLR is asked to compute. Allowable values are:

0 Do not compute flow statistics

1 Compute mean and root mean square (RMS)

-1 Reset previously computed flow statistics and start again

-2 Strip previously computed statistics from the restart file

iaero – Tells DPLR to track integrated aerodynamic variables at each iteration. Allowable values are:

0 Do not track variables

1 Write integrated body forces and moments to file at each iteration

Grid Adjustment / Alignment / Morphing Flags – Each of the flags in this portion of the DPLR input deck gives you a range of options to use if you decide to realign your input grid to better capture the shock wave inside DPLR.

You can accomplish this in two ways:

- Adjusting the values for the grid adaption flags in the DPLR input deck to be used with the restart file after running an initial simulation.
- Creating a runtime control file to adjust the values for the grid adaption flags while the simulation in running. *(See Section 6.4 for more information on Runtime Control Files.)*

To better capture the shock wave using this grid adaption option, you need to:

(1) Move the outer boundary of the input grid to just beyond the shock location as determined by the initial converged solution.

(2) Smooth the outer boundary surface (controlled by the ismooth flag).

(3) Redistribute the interior grid points (controlled by the imradial flag).

`igalign -` Enables and sets the type of grid adaption you want DPLR to use. Allowable values are:

0 Do not perform grid alignment

1 Perform basic grid alignment *(should only be used with a restart file)*

2 Recluster grid only; no alignment *(not dependent upon a shock wave location, can be used in initial simulation but do not use with `imradial=1` setting unless the problem has been previously converged)*

3 Smooth outer boundary only; no alignment *(not dependent upon a shock wave location, can be used in initial simulation)*

5 Perform basic grid alignment, but hold first 40% of grid points in the body-normal direction fixed *(should only be used with a restart file; permits rapid alignment of grid to shock wave and more aggressive CFL ramp between alignments, but should only be used after coarse alignment has been achieved; still experimental) (Not Tested in DPLR 4.01.0)*

11 Automatic grid alignment (basic) *(Allows DPLR to determine when to perform grid adaptions; not extensively tested, so use with caution)*

20 Surface morphing (constant amount) *(Allows you to morph the surface of the body by an amount specified in the* `ds1` *flag; used primarily for surface recession calculations; not extensively tested at this time, so use with caution)*

21 Surface morphing (variable delta specified in a pointwise boundary condition file) *(Not Working in DPLR 4.01.0)*

Tech Tips:

1). *If you set* `igalign=1` *before a restart file from an existing solution exists, a runtime error will occur.*

2). *When* `igalign=5`, `ngeom` *and* `imradial` *are ignored.*

3) *When* `igalign=20`, *the remainder of the flags in this block are ignored and the body-normal distribution is taken as a scaled version of the previous distribution at every body location.*

4). *Two options that were allowed in v3.05,* `igalign=4` *and* `igalign=14`, *are deprecated in this release. As of v4.01.0, the input flag* `ismooth` *is provided in order to give the user more flexibility in controlling the type of smoothing that occurs. The functionality of* `igalign=4` *and* `igalign=14` *can be replicated in the current release by setting* `igalign=1` *and* `igalign=11` *respectively, along with* `ismooth=3`.

`ngiter` - Sets the frequency at which a grid alignment is performed. Recommended values:

500 – 1500

Tech Tip: *The first alignment always occurs on a restart prior to running the first iteration, and subsequent alignments (with the total number set by* `nalign`*) are performed every* `ngiter` *iterations.*

---

i1stadpt - Specifies the multiplier to use to delay the first grid adaption/alignment, which occurs when `nit >= i1stadapt*ngiter`. Default value:

0

imedge - Specifies the method used to locate the bow shock in the simulation. Allowable values are:

1 Align to a constant Mach number contour

Tech Tip: *Although other programs such as SAGe or Outbound have other options, only Mach number-based adaption is supported in DPLR at this time.*

imradial - Specifies the type of wall spacing to employ during the reclustering of interior points that takes place during a grid adaption. Allowable values are:

1 Constant cell Reynolds number wall spacing *(wall spacing varies over the body surface from the value set in* `ds1` *to the value set in* `dslmx`*)*

2 Use a constant wall spacing *(wall spacing at all surface locations will equal the value set in* `ds1`*)*

Tech Tip: *If* `ds1=0`*, the current wall spacing will be used.*

ngeom - Specifies the number of geometrically spaced points to place near the body surface during reclustering. Recommended value:

2

Tech Tip: *If* `ngeom <2`*, a pure two-sided Vinokur stretching routine will be used.*

ismooth - Specifies the type of smoothing to employ at the outer boundary following a grid alignment. Allowable values are:

0 Do not smooth the outer boundary

1 Smooth outer boundary of grid based on changes in arc length at each body point location.

2 Smooth outer boundary of grid based on total final arc length at each body point location.

3 Smooth using both method #1 and method #2. *(preferred for initial adaption from a hyperbolic grid)*

Tech Tips:

***1*).** *Setting* `ismooth=1` *is generally preferred because it tends to give smoother outer boundaries and will eventually asymptote to zero smoothing when the outer boundary stops moving. Also, it avoids some problems seen with SAGe when smoothing grids with abrupt changes in surface geometry, such as a propagation of the surface geometry to the outer boundary, as seen in the Shuttle Orbiter tail region. On the downside, this option will tend to preserve oscillations in the outer boundary if they develop during the solution procedure.*

2). *Setting* `ismooth=2` *is the only approach that can be used for smoothing a grid without adaption (*`igalign=3`*) because all of the arc length changes are zero. Also, this option is preferable for the first adaption when there is an extremely large ratio between the shortest and longest arc lengths in the grid, particularly if short arc lengths occur near a region with high curvature, such as a wing leading edge.*

3)*.Setting* `ismooth=3` *causes the outer boundary of the grid to be smoothed using both algorithms described above, producing superior quality outer boundaries for the initial adaption of hyperbolic grids. Further adaptions should then be peformed with* `ismooth=1`*.*

`fs_scale` - Specifies the fraction of the freestream Mach number to pick as the adaption contour. Recommended values:

$$0.9 \leq fs\_scale \leq 0.95$$

Tech Tip: *Values smaller than 0.90 lead to smoother grids, but increase the chance that the final outer boundary will not contain the entire shock.*

ds_mult - Specifies, as a multiple of the "local" radial grid spacing at the estimated shock location, where to place the realigned outer boundary beyond that location. Must be > 0. Recommended values range:

1.0-3.0, with 2.5 being typical

Tech Tip: *The idea is to leave at least the outer two points of each radial grid line beyond the shock following reconvergence of the flow solution. The shock location tends to move inward with each tailoring, so erring on the low side with ds_mult is normally safe during early adaptions, especially if the initial boundary is far away from the shock.*

gmargin - An optional multiplier of the "outermost" radial spacing of the grid, normally not needed. This permits additional control over the outer grid boundary, and may be used to increase or "decrease" the radial adjustment produced by the normal alignment scheme. Although values may be positive or negative real numbers, the typical value for this input is:

0

Tech Tip: *This control allows the boundary to expand or contract (everywhere) if there is reason to believe the current outer boundary is too close to the eventual shock or too far from it. If extrapolation is implied, the extrapolations are linear. Beware of possible crossed grid lines or excessive cell skewness if any existing radial lines are convergent.*

ds1 - Constant value with different meanings for different settings of `imradial` and `igalign`.

When `imradial=1, ds1` sets the minimum allowable wall spacing anywhere in the volume.

`ds1=0` No minimum allowable wall spacing anywhere in the volume

`ds1<0` Lower limit for cell Reynolds number smoothing *(minimum spacing is limited by* `ds1`*% of the current adapted arc length)*

When `imradial=2, ds1` sets wall spacing everywhere in the volume.

`ds1=0` Maintains wall spacing in the current grid.

`ds1<0` Wall spacing is `ds1`% of the current adapted arc length.

When `igalign=20, ds1` is the constant value by which the surface grid should be morphed in the body normal direction.

`ds1>0` Used for a recession in the surface

`ds1<0` Used for a growth in the surface

`cellRe` - Specifies the value of the cell Reynolds number when `imradial=1`. *(For all other values of* `imradial`*,* `cellRe` *is ignored.)*

`ds1mx` - Specifies the maximum wall spacing allowed when cell Reynolds number spacing is employed `(imradial=1)`. *(For all other values of* `imradial`*,* `ds1mx` *is ignored.)*

`ds1mx<0` Wall spacing is `ds1mx`% of the current adapted arc length.

Tech Tip: *Using arc length-based spacing is not generally recommended, but can be helpful for certain situations, such as the tail root area of the Shuttle orbiter grid.*

`ds2fr` - Specifies the spacing for the outer boundary of the grid. Recommended value:

0.35

Tech Tip: `ds2fr` *is expressed as a fraction of the spacing that would be used if an unconstrained (one-sided) Vinokur stretching algorithm were employed.*

Overset Grid Implementation – The flags in this portion of the DPLR input deck control Overset Grid capabilities of DPLR (i.e., enable Chimera topologies). See Chapter 8 for more information on Using Overset Grids.

Note: DiRTlib and Overset functionality must be available at compile time for this functionality to be enabled.

`iover` – Indicates enabling of Overset Logic, if compiled, in DPLR. Allowable values are:

- 0 Overset logic is disabled
- 1 Overset logic is enabled

`ioint` – Indicates format of domain connectivity file (cname). Allowable values are:

- 0 ASCII Suggar-type .dci output

`xxxxx` – Not used in DPLR 4.01.1 (reserved for future use)

Block-Specific Flags – The flags in this portion of the DPLR input deck can be set differently for each computational block in the simulation.

`ntx, nty, ntz` – Specifies the total number of computational cells in the i, j, k directions for a block. Should be set to the number of interior cells, not including dummy or ghost cells. *(Note that ntz is only used for 3D flow simulations.)*

`iconr` – Specifies how to initialize this block for simulation when then global modeling flag `iinit=10`. Allowable values are:

- 0 Start by initializing to the values in the freestream specification identified in `ifree`
- 1 Restart from saved file
- 2 Start with a stagnant interior at low pressure
- 3 Start with artificial boundary layer in place

isim – Includes or excludes a master block from the simulation. Allowable values are:

0 Do not include this block in the simulation

1 Include this block in the simulation

Tech Tip: *Because excluding blocks does not save on the computational intensity of the simulation, this setting is only used when you want to freeze problem blocks while the remainder of the solution is allowed to converge.*

ifree - Identifies the number of the freestream specification to use for this block.

initi - Identifies the number of the freestream specification to use to initialize the interior of this block.

ibadpt - Indicates whether grid adaption will be applied to a specific block. Allowable values are:

0 Do not perform adaption on this block *(Not Working in DPLR 4.01.1)*

1 Perform adaption on this block

(ijk)flx - Specifies the method to use to extrapolate the Euler fluxes in the *i, j*, or *k* directions. Note that the method for flux extrapolation can be set separately in each computational direction. Allowable values are:

0 No flux evaluation

1 Upwind modified Steger-Warming with Δp

2 MUSCL Steger-Warming with Δp [p, c_s, $\vec{u}$, $\vec{T}$] *(recommended value)*

3 MUSCL Steger-Warming with Δp [ρ_s, $\vec{u}$, $\vec{T}$]

4 MUSCL Steger-Warming with Δp [p, c_s, $\vec{u}$, $\vec{e}_i$, T]

5 Pure 2nd order central difference *(should not be used for problems which contain shock waves; could be unstable even for subsonic flows.)*

11 Upwind modified Steger-Warming without Δp

12 MUSCL Steger-Warming without Δp [p, c_s, $\vec{u}$, $\vec{T}$]

13 MUSCL Steger-Warming without Δp [ρ_s, $\vec{u}$, $\vec{T}$]

14 MUSCL Steger-Warming without Δp [p, c_s, $\vec{u}$, $\vec{e}_i$, T]

Tech Tip: *The settings* `(ijk)flux` *= 2-4 and 12-14 use a MUSCL-based adaptive stencil to attain higher-order accuracy via a more sophisticated approach. The difference between the selections is in the set of variables that are extrapolated to attain high-order accuracy, and whether a pressure gradient based switch is employed to smoothly transition from high order to first-order in the vicinity of strong shock waves.*

`(ijk)ord` – Specifies the nominal order of accuracy of the Euler flux extrapolation. Allowable values are:

1 First-order upwind

2 Second-order upwind biased

3 Third-order upwind biased *(recommended value)*

`omg(ijk)` – Specifies the value of ω (as defined by Yee) to employ in the MUSCL extrapolation scheme. Recommended value:

3

Tech Tip: *DPLR will automatically reset this value to 2 for second-order simulations.*

`(ijk)lim` – Specifies the type of flux limiter to use in the Euler flux extrapolation. Allowable values are:

1 Minmod *(recommended value)*

2 Superbee

3 Van-Albada

Tech Tip: *The Superbee and Van-Albada flux limiters, while somewhat less dissipative, are also less stable and should only be used when low dissipation schemes are actually necessary to obtain highly accurate solutions, such as reactive mixing layer flows.*

`(ijk)diss` – Specifies the type of eigenvalue limiter to use in the Euler flux extrapolation. Allowable values are:

0 No added dissipation *(recommended value for body normal direction)*

1 Standard eigenvalue limiting *(recommended value for radial and circumferential directions)*

2 Standard eigenvalue limiting on linear fields only

3 Standard eigenvalue limiting on non-linear fields only

Tech Tip: *Normally, eigenvalue limiters should be used in the radial and circumferential flow directions, but should be avoided in the body-normal direction when possible to avoid adding dissipation in the boundary layer. However, in those very rare instances when a normal direction limiter can be helpful, set* `(ijk)diss=3` *to apply the limiter only to fluxes with non-linear eigenvalues or set* `kbl` *to a positive integer to turn off the eigenvalue limiter within the boundary layer.*

`eps(ijk)` – Specifies the magnitude of the eigenvalue limiter to use in the Euler flux extrapolation. Recommended values for hypersonic blunt body flow simulations are:

0 No added dissipation – Recommended for body normal direction

0.3 Recommended for radial and circumferential directions

Tech Tip: *Much lower values of* `eps(ijk)` *– on the order of 0.01- can be used in separated flows which have no strong shocks and are much more sensitive to the effects of added (artificial) dissipation. DPLR will print a run-time warning if it detects a non-zero value of* `eps(ijk)` *in the body-normal direction in any block.*

`iextst` – Specifies the time advancement method to use in the simulation. Allowable values are:

1 Explicit first-order Euler

2 Explicit second-order Runge-Kutta

-1 Implicit data-parallel line relation (DPLR) *(recommended value for steady-state problems)*

-2 Implicit data-parallel full matrix (FMDP)

Tech Tip: *For time accurate calculations, only the relatively inefficient second-order Runge-Kutta (Midpoint) method is offered at this time.*

nrlx – Specifies the number of implicit relaxation steps to use when using the DPLR or FMDP time advancement methods (`iextst= -1 or -2`). Recommended value:

4

ildir – Specifies the direction in which the lines are to be formed for the DPLR time advancement method (`iextst= -1`). Allowable values are:

0 Autodetect direction *(recommended value)*

1 *i*-direction

2 *j*-direction

3 *k*-direction

4 Alternate directions *(changes orientation of lines with each iteration, alternating between i. j, and k direction solves. Provided for use with separated flows, although no significant advantage has been shown with this method.)*

Tech Tips:

1). *The DPLR time advancement method is based on the Gauss-Seidel Line Relaxation (GSLR) method, and the lines should be formed in the body-normal direction for maximum performance. Setting* `ildir = 1, 2, or 3` *will cause the code to orient the solver in that block so that lines are formed in the i, j, or k-directions respectively. If DPLR detects that a line is formed in a non body-normal orientation a warning message will be printed. It is not a fatal error to run DPLR with the lines in non body-normal directions, but the convergence rate and stability of the method will be degraded.*

2) *When* `ildir=0`*, DPLR will automatically determine the best direction to form the lines for each block at runtime by examining the block boundary conditions. For those blocks for which no body-surface boundary condition is detected, the lines will be formed in the i-direction. For those blocks with a body surface boundary condition at more than one face, the lines will*

be formed in the direction normal to the last body-surface detected.

ibcu – Specifies how often to update the implicit boundary conditions during the relaxation process for DPLR or FMDP (`iextst = -1 or -2`). Recommended value:

1

Tech Tip: *Updating implicit boundary conditions improves parallel efficiency on machines for which message-passing is very inefficient.* `ibcu=1` *forces the implicit boundary conditions to be updated during each relaxation step.*

iblag – Specifies whether to lag the implicit boundary conditions when using DPLR or FMDP (`iextst = -1 or -2`). Allowable values are:

-1 or 0 Do not lag implicit boundary conditions *(recommended value)*

1 Lag implicit boundary conditions

Tech Tip: *Although it is generally desirable to lag the implicit boundary condition update in order to better mask the message-passing latency and improve parallel performance of the time advancement method, there are certain instances when the block topology employed makes lagged boundary conditions dangerous. In the future, DPLR may automatically determine whether latency can be masked for a given application and this flag will be automatically set by the code.*

ilt – Specifies whether to employ global or local time stepping for implicit simulations. Allowable values are:

-1 Global time stepping

-2 Global time stepping with maximum CFL limit

1 Local time stepping

2 Local time stepping with a maximum CFL limit

Tech Tips:
1). *When the DPLR method of time stepping is chosen* `(iextst=-1)`*, only global time stepping* `(ilt=-1 or -2)` *should be used.*

2) *When one or more blocks of a complex simulation are unstable, you can specify a maximum CFL number to use only for the problem blocks by setting* `il =+2` *and entering a maximum CFL number in the* `cflm` *flag.*

`ibdir` – Specifies the grid direction in which to break single-block problems for parallel execution. Allowable values are:

1 *i*-direction
2 *j*-direction
3 *k*-direction

Tech Tip: *When the simulations has only one master block with no zonal interfaces, DPLR can perform the necessary decomposition at runtime by breaking the problem into planes in the direction chosen with the* `ibdir` *flag. Note that DPLR will print a warning message if* `ibdir` *is set such that the grid is broken in the body-normal direction.*

`cflm` - Specifies the maximum CFL number to use in the current master block. *(Only used when* `ilt = + 2`*).*

`ibc` – Specifies the type of boundary condition to use at each of the six faces of each master block, i.e., **`imin, imax, jmin, jmax, kmin, kmax`**. Allowable values for each face are:

Basic Boundaries: 0-29
---

0 Pointwise boundary condition read from input "*.pbca" file
1 Fixed at freestream conditions *(specified by* `ifree`*)*
2 Fixed at freestream if inflow; extrapolate if outflow *(used in rapid analysis process)*

3	First order extrapolation *(supersonic exit)*	
4	Second order extrapolation *(supersonic exit)*	
5	RESERVED	
6	Subsonic reservoir inlet; constant mass flow	
7	Periodic	
8	Inviscid wall *(flow tangency)*	
9	Viscous adiabatic wall	
10	Viscous isothermal wall	
11	180 degree singular axis ($u = -u$)	[3D]
12	180 degree singular axis ($v = -v$)	[3D]
13	180 degree singular axis ($w = -w$)	[3D]
14	Singular x-axis ($v = -v$)	[axi]
15	Singular y-axis ($u = -u$)	[axi]
16	360 degree singular axis	[3D]
17	Plane of symmetry ($u = -u$)	
18	Plane of symmetry ($v = -v$)	
19	Plane of symmetry ($w = -w$)	
20	Zone boundary	
21	90 degree singular axis ($v=-v;\ w=-w$)	[3D]
22	90 degree singular axis ($u=-u;\ w=-w$)	[3D]
23	90 degree singular axis ($u=-u;\ v=-v$)	[3D]
24	RESERVED	
25	Catalytic isothermal wall	
26	Catalytic radiative equilibrium wall	
27	Non-catalytic radiative equilibrium wall	

Blowing Wall Boundaries: 30 - 39

30	Viscous isothermal wall with blowing
35	Catalytic isothermal wall with blowing *(Not Working in DPLR 4.01.1)*
36	Catalytic radiative equilibrium wall with blowing *(Not Working in DPLR 4.01.1)*

37 Non-catalytic radiative equilibrium wall with blowing *(Not Working in DPLR 4.01.1)*

38 inviscid wall with blowing *(Not Working in DPLR 4.01.1)*

39 viscous adiabatic wall with blowing *(Not Working in DPLR 4.01.1)*

Slip Wall Boundaries: 40 - 49

---

40 Viscous isothermal wall with slip

45 Catalytic isothermal wall with slip *(Not Working in DPLR 4.01.1)*

46 Catalytic radiative equilibrium wall with slip *(Not Working in DPLR 4.01.1)*

47 Non-catalytic radiative equilibrium wall with slip *(Not Working in DPLR 4.01.1)*

49 Viscous adiabatic wall with slip

Input Profile Boundaries: 60 - 69

---

60 Input primitive variables (ρ_s, *u, v, w, Tv, T*)

61 Input primitive variables (p, c_s [2-ns], *u, v, w, Tv, T*)

62 Input conserved variables (ρ_s, ρu, ρv, ρw, *Ev, E)*

Tech Tip: *Entering one of these numbers will tell DPLR to look in the "*.pcba" file to find values for the indicated variables. If such a file does not exist, a runtime error may occur.*

Material Response Coupling Boundaries: 70 - 79

---

70 Input species *mass flow rate* & *T,* extrapolate for *p* with thermal equilibrium assumed

71 Input c_s, *mass flow rate* & *T*, extrapolate for *p* with thermal equilibrium assumed

75	Activate surface kinetic mechanism, isothermal (`icatmd=1001`)
76	Activate surface kinetic mechanism, radiative equilibrium (`icatmd=1001`)

Subsonic Inflow/Outflow Boundaries: 80 - 89

81	Subsonic reservoir inlet; same as #6
82	Subsonic inlet; specify mass flow rate (`density * M/ReV` (normal velocity)) & *T*, extrapolate *p*
85	Subsonic exit; specify static pressure (`pback`), extrapolate others
86	Subsonic inlet; specify subsonic temperature (`subT0`) and subsonic pressure (`subp0`); Assumed isentropic and T=Tv. Uses methods of characteristics.
87	Subsonic exit; specify static pressure (`pback`), extrapolate others. Uses method of characteristics. Uses methods of characteristics.
88	Subsonic exit; specify static pressure (`pback`), extrapolate others. Uses method of characteristics, disallows backflow.

Pointwise Twall Boundaries: 100 – 199 *(100 + corresponding isothermal boundary condition number)*

110	No slip (viscous) isothermal wall
125	Catalytic isothermal wall
130	No slip (viscous) isothermal wall with blowing
135	Catalytic isothermal wall with blowing (*Not Working in DPLR 4.01.1*)
140	Isothermal wall with slip
145	Catalytic isothermal wall with slip (*Not Working in DPLR 4.01.1*)

Pointwise Twall and Blowing Boundaries: 200 – 299 *(200 + corresponding non-blowing boundary condition number)*

230	No slip (viscous) isothermal wall
235	Catalytic isothermal wall (*Not Working in DPLR 4.01.1*)

Pointwise Blowing Boundaries: 300 – 399 *(300 + corresponding non-blowing boundary condition number)*

330	No slip (viscous) isothermal wall with blowing
335	Catalytic isothermal wall (*Not Working in DPLR 4.01.1*)

Tech Tip: *To specify a pointwise boundary for any cell face, you must first set up and include a "*.pbca" with the appropriate data in your current working directory.*

Overset Grid Boundary: 900 - 999

901	Specifies an overset boundary (supersonic exit 1st order extrapolation if `iover=0`)

Mathematically Adjusted Boundaries: 1000 – 1099
(1011 – 1019 & 1021-1023 currently support augmented eigenvalue limiters in the vicinity of standard singular axes or symmetry planes when `kdg` *≠ 0.)*

1011	180 degree singular axis ($u = -u$)	[3D]
1012	180 degree singular axis ($v = -v$)	[3D]
1013	180 degree singular axis ($w = -w$)	[3D]
1014	Singular x-axis ($v = -v$)	[axi]
1015	Singular y-axis ($u = -u$)	[axi]
1016	360 degree singular axis	[3D]
1017	Plane of symmetry ($u = -u$)	
1018	Plane of symmetry ($v = -v$)	

1019	Plane of symmetry ($w = -w$)
1021	90 degree singular axis ($v=-v$; $w=-w$) [3D]
1022	90 degree singular axis ($u=-u$; $w=-w$) [3D]
1023	90 degree singular axis ($u=-u$; $v=-v$) [3D]

Mathematically Adjusted Boundaries: 2000 – 2099
(2011 – 2019 & 2021-2023 currently support a maximum CFL in the vicinity of standard singular axes axes or symmetry planes as per `cflm` *in each block when* `kdg` $\neq 0$.)

2011	180 degree singular axis ($u = -u$)	[3D]
2012	180 degree singular axis ($v = -v$)	[3D]
2013	180 degree singular axis ($w = -w$)	[3D]
2014	Singular x-axis ($v = -v$)	[axi]
2015	Singular y-axis ($u = -u$)	[axi]
2016	360 degree singular axis	[3D]
2017	Plane of symmetry ($u = -u$)	
2018	Plane of symmetry ($v = -v$)	
2019	Plane of symmetry ($w = -w$)	
2021	90 degree singular axis ($v=-v$; $w=-w$) [3D]	
2022	90 degree singular axis ($u=-u$; $w=-w$) [3D]	
2023	90 degree singular axis ($u=-u$; $v=-v$) [3D]	

Freestream Specification Flags – The flags in this portion of the DPLR input deck define a set of conditions for an area of the flow from which all relevant fluid dynamic quantities can be computed. You can define any number of freestream areas this way (the total given in `nfree`) and use them to initialize master blocks and set freestream boundary conditions on a block-by-block basis using the `initi` and `ifree` flags, respectively. *(In DPLR, freestream values are always expressed in standard international (SI) units.)*

`irm` – Specifies whether a velocity, Mach number, or unit Reynolds number will be given as input. Allowable values are:

1	Mach number (*assumed to be the equilibrium - as opposed to frozen - value)*
2	Reynolds number per meter

3 Velocity *(recommended value)*

Tech Tip: *The most common (and least ambiguous) input for most free-flight simulations is velocity, because each of the other entries requires the velocity to be derived from the thermodynamic and transport models employed in the given simulation.*

density - Specifies the input freestream mass density.

M/Re/V - Specifies the input Mach number, unit Reynolds number, or velocity depending upon the value of `irm`.

Tech Tip: *Whichever choice is made in `irm`, DPLR will determine the remaining two values using the input thermodynamic and transport models distributed with the code.*

c(xyz) - Specifies the input velocity vector direction cosine. Allowable values are:

```
0.0 < value < 1.0
```

Tech Tip: *These values must be nondimensionalized.*

Tin, Tvin, Trin, Tein - Specifies the input translational, vibrational, rotational, and free electron temperatures, respectively.

Tech Tips:
1) The number of unique temperatures depends upon the thermal non-equilibrium models chosen with the `ivib`, `irot`, `ieex`, and `iel` flags.

2) At the current time, free electron non-equilibrium is not supported in DPLR and so will be silently ignored.

3) Because only one thermal non-equilibrium model may be employed in a given simulation, all blocks are assumed to be governed by the same model. However, each block can have different initial or freestream temperatures by defining multiple

freestream specifications and using the `initi` *and* `ifree` *flags in each block characterization.*

turbi - Specifies a freestream turbulence level for the two-equation turbulence models. For SST, specify omega as follows:

- > 0 use directly to define turbulence level
- ≤ 0 set turbulence level to default level of 1.0E-4

Tech Tip: *Not used for laminar or Baldwin-Lomax (algebraic) turbulent simulations.*

tkref - Turbulent viscosity ratio. Initializes the freestream value of turbulent viscosity for Spalart-Allmaras turbulence models (`itmod = 1000+`*n*). Recommended value = 0:

- `0` *(freestream turbulent viscosity specified by* $\mu_T = 0.1 * \mu_L$ *)*
- `>0` *(freestream turbulent viscosity specified by* $\mu_{T\infty} = tkref$ *)*
- `<0` *(freestream turbulent viscosity specified by* $\mu_{T\infty} = |tkref|\mu_{L\infty}$ *)*

Tech Tip *Not used for laminar or Baldwin-Lomax (algebraic) turbulent simulations.*

subp0 – Specifies stagnation pressure in simulations where subsonic boundary conditions are identified.

subT0 - Specifies stagnation temperature in simulations where subsonic boundary conditions are identified.

pback - Specifies back pressure for subsonic outflow in simulations where subsonic boundary conditions are identified..

cs - An array of input species mass fractions.

Tech Tips:
1) *There must be one entry per species in the chosen chemistry model as specified in the input *.chem file.*
2) *All input mass fractions must sum to 1.0 or DPLR will exit with an error message.*
3) *Input of mole fractions is not supported at this time.*

CFL Number Listing – The final entries in the DPLR input deck are a list of Courant-Friedrichs-Lewy (CFL) numbers to employ during the simulation.

CFL numbers are a measure of the explicit inviscid stability limited time step Δt and are used by convention in CFD codes to enable time advancement to a steady state solution. In DPLR, the CFL number for a given computational cell is defined as the time it takes the fastest wave in the flow to traverse the thinnest dimension of the cell.

Global Timestepping: For implicit (non-time accurate) simulations, this time step is a bit different for every cell in the flow. However, most DPLR-based simulations use the minimum value of Δt at any cell in the flowfield for all cells in what is called *global timestepping* – a set-up approach that has been shown to result in robust solutions and good convergence rates.

Local Timestepping: DPLR does offer you the ability to implement *local timestepping* where the local value of Δt is used for each computational cell by setting `ilt=1,2`, but this approach is recommended only for simulations using the full-matrix data-parallel method (FMDP), i.e., `iextst=-2.`

CFL Number Ranges: When you begin a simulation, you should use a very small CFL value to verify that your proposed solution will, indeed, converge. After several hundred iterations, if you see that the solution is progressing toward convergence, you can increase or "ramp" the CFL value to specify a larger timestep and speed up the solution process. Between each listed CFL number, DPLR will perform 20 iterations of the solution. If you add an integer to a line with a CFL number, DPLR will perform 20-times-that-integer iterations. *(Note: After the first grid adaption, DPLR performs only 10 iterations between CFL numbers and 10-times-the-added-integer iteration on the assumption that a post-adaption grid is a better "starting point" and justifies more aggressive CFL ramping.)*

For example, the following CFL number listing might be appropriate for an initial simulation of a problem:

```
0.01          DPLR performs 20 iterations at a 0.01 timestep
0.05          DPLR performs 20 iterations at a 0.05 timestep
0.10      2   DPLR performs 40 iterations at a 0.10 timestep
```

0.15	3	DPLR performs 60 iterations at a 0.15 timestep
0.25	3	DPLR performs 60 iterations at a 0.25 timestep
0.50	3	DPLR performs 60 iterations at a 0.50 timestep
1.0	5	DPLR performs 100 iterations at a 1.0 timestep
2.0	5	DPLR performs 100 iterations at a 2.0 timestep
5.0	5	DPLR performs 100 iterations at a 5.0 timestep
10.0	5	DPLR performs 100 iterations at a 10.0 timestep
20.0		DPLR performs 20 iterations at a 20.0 timestep
50.0		DPLR performs 20 iterations at a 50.0 timestep
100.0		DPLR performs 20 iterations at a 100.0 timestep
250.0		DPLR performs 20 iterations at a 250.0 timestep
500.0		DPLR performs 20 iterations at a 500.0 timestep
1,000.0		DPLR performs 20 iterations at a 1,000.0 timestep
-1		No more CFL values are available

Although larger CFL numbers imply larger timesteps and faster convergence rates, there is usually a maximum CFL number that represents a stability limit for a given problem. Using CFL values above this number can result in solution divergence. For optimum performance, therefore, you should run at CFL numbers that approach, but do not exceed this limit.

Over time and with experience, you will notice that certain classes of problems are associated with an approximate range of stable CFL numbers, like those listed above that were used for a capsule-shaped problem. By starting your DPLR run with one of these stable CFL ranges, you should be able to get close enough to a solution to create a restart file that can then be further customized by editing the CFL number range in the input deck.

Tech Tips:
1) *You can adjust the CFL number during a DPLR run by using a runtime control (*.ctrl) file. See Section 6.4.*

2) *You can use exact timesteps (*Δt*) instead of CFL numbers by entering negative numbers into the CFL Number Listing area of the DPLR Input Deck, however you cannot use both CFL numbers* and Δt *values in a single DPLR simulation. Note that the term -1 in the CFL number list tells DPLR to stop reading the CFL number list and refer to the* `istop` *flag for the final iteration number.*

4.3 'Neptune' Sample Case

The sample case used throughout the DPLR Code User Manual to illustrate how the Code Package works describes a Neptune entry type probe with an ellipsoidal body as

shown in Figure 4.1. This case is an example of aerocapture, where drag from the atmosphere is used to decelerate the vehicle and bring it into orbit.

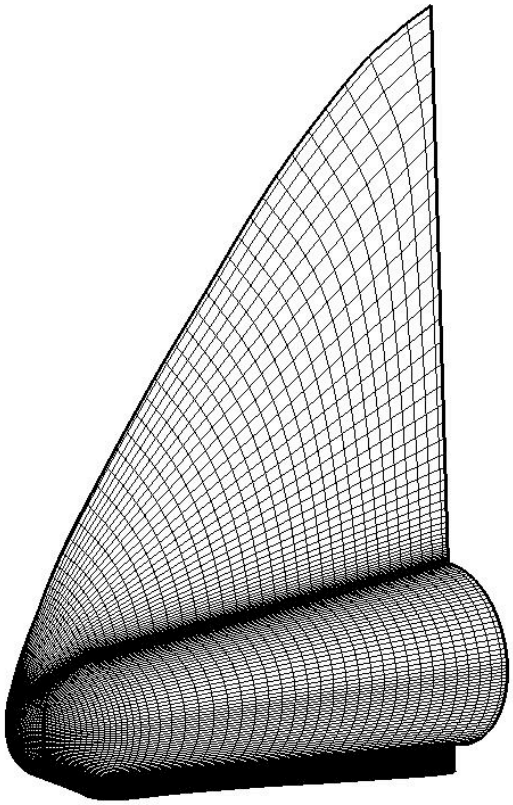


Figure 4.2 – Neptune Probe

4.3.1 Neptune DPLR Input Deck

The DPLR input deck below shows the problem-specific entries made before running the initial DPLR run was made.

```
INPUT DECK FOR DPLR2D/DPLR3D CODE

gname,fname,bname,rname,dname,cname
'neptune-8PE'
'neptune'
'none'
'none'
‘none’
' sde1-fs/twhite/SF/dpcodeV4-00-0/cfdinput/neptune5sp_leibowitz76.chem'

     nblk     igrid     irest      ibcf     iradf     nfree     iinit
       2,        11,       11,        0,        0,        1,        -1

     ivis       ikt       ikv     ivmod     idmod     itmod     islip  iblow
       1,        1,        11,       3,        1,        0,        0,     -1,

   icatmd    ireqmd     twall      epsr    gamcat     xxxxx     vwall
      2,       101,    3.0d3,    0.85d0,   1.0d0,    1.01d5,      0

    ichem      ikeq      ivib      irot      ieex       iel      irad   ipen
       1,        3,        2,        2,        0,        1,        0,     0,

   itrmod    itrans     trloc     trext     itshk
      0,         0,     1.0d0,    0.1d0,       0,

    istop     nplot     iplot      iaxi      ires
    500       100,        1         -1        -2

    igdum       kbl       kdg    istate     iresv
       1,        0,        0,        0,        1,

   xscale       ils     Le/Sc   LeT/ScT      prtl     prtlT
   1.0d0,        2,      .5d0,    1.00d0,    0.72d0   0.90d0,

   xxxxx      xxxx       rvr    resmin
   0.0d0      1.0d0     1.3d0,  1.0d-20


==========================================
Space Marching 1D Implementation
==========================================

   ispace    dxmin      slength    nxtot
      0      1.0d-5      1.0d0      1000


==========================================
TIME ACCURATE & STATISTICAL OPTIONS
==========================================

 itime  lmax      dttol      tfinal      tfac
   0      5      1.0d-3      9.0d99      1.0d15

 ifstat iaero
   0      0
```

```
================================================================
 GRID ADJUSTMENT/ALIGNMENT/MORPHING
================================================================

  igalign    ngiter    nalign
   0,         500,         1,

   imedge  imradial     ngeom     ismooth
       1,     2,          2,        3,

  fs_scale   ds_mult   gmargin
   0.95,      2.5,      0.0,

      ds1    cellRe     ds1mx     ds2fr
     0.0,    1.0,       1.0d-4,    0.3


================================================================
 OVERSET GRID IMPLEMENTATION
================================================================

   iover     ioint    xxxxx
    0         1        0


================================================================================
 BLOCK #1
================================================================================

      ntx       nty       ntz     iconr     isim    ifree    initi    ibadpt
       32,       16,        64,     -1,        1,       1,         1,         1

     iflx      iord      omgi      ilim     idiss      epsi
       4,         3,      2.0d0,       1,         1,       0.3,

     jflx      jord      omgj      jlim     jdiss      epsj
       4,         3,      2.0d0,       1,         1,         0.3,

     kflx      kord      omgk      klim     kdiss      epsk
       4,         3,      2.0d0,       1,         0,       0.03,

   iextst      nrlx     ildir      ibcu     iblag       ilt    ibdir   cflm
     -1,         4,        0,         1,       -1,        -1,    1,     1.0d20

  Boundary condition type [ibc]:
    imin imax jmin jmax kmin kmax
    20,   20,   20,  19,  26,   1

================================================================================
 BLOCK #2
================================================================================

      ntx       nty       ntz     iconr    isim    ifree    initi    ibadpt
       48,       64,       64,     -1,       1,     1,       1,         1

     iflx      iord      omgi      ilim     idiss      epsi
       4,         3,      2.0d0,       1,         1,       0.3,

     jflx      jord      omgj      jlim     jdiss      epsj
```

```
      4,        3,      2.0d0,       1,        1,       0.3,

     kflx      kord      omgk      klim     kdiss      epsk
      4,        3,      2.0d0,       1,        0,       0.03,

   iextst      nrlx     ildir      ibcu     iblag       ilt    ibdir   cflm
     -1,         4,       0,         1,       -1,        -1,    1,    1.0d20

  Boundary condition type [ibc]:
    imin imax jmin jmax kmin kmax
    20,   3,    19,  19,  26,   1

=========================================================================
 Freestream Specification #1
=========================================================================

     irm    density        M/Re/V     cx              cy           cz
      3,    1.6313d-5      3.1045d4, 0.8090160044,  0.5877852523, 0.0d0

     Tin      Trin       Tvin      Tein
   140.3,    140.3,    140.3,    140.3,

   turbi     tkref
   0.001d0,  0.00d0

   subp0      subT0      pback
   2.650d2   2.650d2    1.05d5

      cs        (Species order: H2  H  He)
0.6822392
0
0
0.3177608
0

================================================================
 List of CFL numbers or timesteps for ramping
================================================================

.00001
.0001
.001
.01
.1
1
5
10
20
50
100
200
500
750
1000
2000
5000
-1
```

Figure 4-3 DPLR Input Deck for Neptune Probe

4.3.2 Neptune DPLR Input Deck Settings

This is a three-dimensional problem with one plane of geometric symmetry. The original grid consists of two master blocks. The following table explains the meaning of the DPLR input deck settings in this sample case.

Global Flags	Setting	Explanation
nblk	2	There are 2 master grid blocks in this simulation.
igrid	11	The input grid file is a parallel archival XDR file.
irest	11	The restart file to be created from this simulation will be a parallel archival XDR file.
ibcf	0	Do not read a boundary condition file.
iradf	0	Do not read a radiation file.
nfree	1	There is one region of the freestream (a.k.a. freestream specification) characterized in this DPLR input deck.
iinit	-1	Start all blocks by initializing to the values in the freestream specification characterized in this DPLR input deck.
ivis	1	DPLR will perform a laminar, full Navier-Stokes simulation.
ikt	1	Translational thermal conductivity is modeled in a manner consistent with the baseline model used to compute mixture viscosity and thermal conductivity, specified in the `ivmod` flag.
ikv	11	Vibrational thermal conductivity is modeled with standard expression with T_v gradients.
ivmod	3	The baseline model used to compute mixture viscosity and thermal conductivity is the Yos approximate mixing rules which is preferred for all reacting gas flows.
idmod	1	The species diffusion coefficients for this simulation are computed with a constant Lewis/Schmidt number.
itmod	0	This value is ignored because `ivis`=1, defining the problem as a laminar flow simulation and telling DPLR to ignore turbulence- and transition-related flags.
islip	0	Slip-wall calculations will be disabled.
iblow	-1	Blowing-wall calculations will be disabled.

Global Flags (cont.)	Setting (cont.)	Explanation (cont.)
icatmd	2	Wall catalysis is calculated with the constant γ, fully catalytic to ions but supports homogeneous surface reactions such as $N + N = N_2$ & $O + O = O_2$.
ireqmd	101	The radiative equilibrium wall is modeled with constant wall emissivity set by the value in `epsr` and a maximum wall temperature set by value in `twall`.
twall	3.0d3	Maximum temperature at the vehicle surface = 3,000 degrees Kelvin.
epsr	0.85d0	The surface material is 85% efficient in emitting energy away from the vehicle.
gamcat	1.0d0	The value of γ for the constant γ homogeneous catalysis model is 1.
xxxxx	1.0d5	This flag is currently ignored in DPLR.
vwall	0	This value is ignored because `iblow`=-1, telling DPLR to disable blowing-wall calculations.
ichem	1	Finite-rate chemistry is employed for the chemical reactions in the gas phase.
ikeq	3	The equilibrium constants are computed from the Park 1990 model ($n = 10^{16}/cm^3$).
ivib	2	Vibrational energy is computed with vibrational equilibrium using statistical mechanics.
irot	2	Rotational energy is computed with rotational equilibrium using statistical mechanics.
ieex	0	Electronic energy of the gas is not modeled.
iel	1	Free electron energy of the gas is computed using the coupled free electron and translational energy model.
irad	0	No model is used to compute shock layer radiation.
ipen	0	Not used by DPLR at this time.
itrmod	0	This value is ignored because `ivis`=1, defining the problem as a laminar flow simulation and telling DPLR to ignore turbulence- and transition-related flags.

Global Flags (cont.)	Setting (cont.)	Explanation (cont.)
itrans	0	This value is ignored because `ivis`=1, defining the problem as a laminar flow simulation and telling DPLR to ignore turbulence- and transition-related flags.
trloc	1.0d0	This value is ignored because `ivis`=1, defining the problem as a laminar flow simulation and telling DPLR to ignore turbulence- and transition-related flags.
trext	0.1d0	This value is ignored because `ivis`=1, defining the problem as a laminar flow simulation and telling DPLR to ignore turbulence- and transition-related flags.
itshk	0	Not used by DPLR at this time.
istop	500	DPLR will run 500 iterations before stopping the simulation.
nplot	100	DPLR will write a restart file every 100 iterations.
iplot	1	DPLR will save only the most recently written restart file.
iaxi	-1	This is a non-axisymmetric simulation.
ires	-2	Screen output for this simulation will include the iteration number, the global residual, and the CFL number being used and will include a comparison with these values from the first iteration of the simulation.
igdum	1	DPLR will compute grid dummy cell coordinates.
kbl	0	DPLR will ignore this flag.
kdg	0	DPLR will ignore this flag
istate	0	DPLR will use the equation of state for a perfect gas.
iresv	1	DPLR will track the total density of all species in the simulation.
xscale	1.0d0	DPLR will not scale the input grid at runtime.
ils	2	Input numbers governing laminar diffusion coefficients will be interpreted as Schmidt Numbers.
Le/Sc	0.5d0	The Schmidt number to be used in the simulation is 0.5.
LeT/ScT	1.00d0	This value is ignored because `ivis`=1, defining the problem as a laminar flow simulation and telling DPLR to ignore turbulence- and transition-related flags.
prtl	0.72d0	This value is ignored because `ivmod=3` (not 2 or 12).

Global Flags (cont.)	Setting (cont.)	Explanation (cont.)
prtlT	0.90d0	This value is ignored because `ivis`=1, defining the problem as a laminar flow simulation and telling DPLR to ignore turbulence- and transition-related flags.
xxxx	0.0d0	Not used by DPLR at this time.
xxxx	1.0d0	Not used by DPLR at this time.
rvr	1.3d0	The viscous overrelaxation parameter for this simulation is 1.3.
resmin	1.0d-20	When this simulation converges into a solution, the residual will be essentially zero.

Space Marching 1D Impementation Flags	Setting	Explanation
ispace	0	Space marching is disabled in this simulation.
dxmin	1.0d-5	This value is ignored because `ispace` = 0.
slength	1.0d0	This value is ignored because `ispace` = 0.
nxtot	1000	This value is ignored because `ispace` = 0.

Time Accurate & Statistical Options Flags	Setting	Explanation
itime	0	Use a 1st order integration of time accuracy.
1max	5	This value is ignored because itime ≠ 1.
dttol	1.0d-3	This value is ignored because itime ≠ 1.
tfinal	9.0d99	Final flow time is essentially infinity.
tfac	1.0d15	This value is ignored because itime ≠ 1.
ifstat	0	Do not compute flow statistics.
iaero	0	Do not compute aerodynamic variables.

Grid Adjustment / Alignment/ Morphing Flags	Setting	Explanation
igalign	0	Grid alignment will not be performed in this simulation.
ngiter	500	This value is ignored because `igalign=0`.
nalign	1	This value is ignored because `igalign=0`.
imedge	1	This value is ignored because `igalign=0`.
imradial	2	This value is ignored because `igalign=0`.
ngeom	2	This value is ignored because `igalign=0`.
ismooth	3	This value is ignored because igalign=0.
fs_scale	0.95	This value is ignored because `igalign=0`.
ds_mult	2.5	This value is ignored because `igalign=0`.
gmargin	0.0	The outermost radial spacing of the grid will remain as specified in the grid file.
dsl	0.0	This value is ignored because `igalign=0`.
cellRe	1.0	This value is ignored because `igalign=0`.
dslmx	1.0d-4	This value is ignored because `igalign=0`.
ds2fr	0.3	This value is ignored because `igalign=0`.

Overset Grid Implementation Flags	Setting	Explanation
iover	0	Overset logic is disabled for this simulation.
ioint	1	This value is ignored becaue `iover=0`.
xxxx	1	This value is not used in DPLR 4.01.0.

Block #1 Flags	Setting	Explanation
ntx	32	There are 32 computational cells in the *x* direction in Block #1.
nty	16	There are 16 computational cells in the *y* direction in Block #1.
ntz	64	There are 64 computational cells in the *z* direction in Block #1.
iconr	-1	This value is ignored because `iinit=-1`.
isim	1	This block will be included in the simulation.
ifree	1	Use freestream specification #1 for this master block.
initi	1	Use freestream specification #1 to initialize the interior of this master block.
ibadpt	1	Grid adaption will be performed on this block.
iflx	4	The Euler flux extrapolation method to use in the *i* direction is MUSCL Steger-Warming with Δp.
iord	3	The Euler flux extrapolation order of accuracy is third-order upwind biased.
omgi	2.0d0	The value of ω to employ in the MUSCL scheme is 2.
ilim	1	The MinMod flux limiter is used in the Euler flux extrapolation.
idiss	1	A standard eigenvalue limiter is used in the flux extrapolation.
epsi	0.3	The magnitude of the eigenvalue limiter is 0.3 in the flow direction
jflx	4	The Euler flux extrapolation method to use in the *j* direction is MUSCL Steger-Warming with Δp.
jord	3	The Euler flux extrapolation order of accuracy is third-order upwind biased.
omgj	2.0d0	The value of ω to employ in the MUSCL scheme is 2.
jlim	1	The MinMod flux limiter is used in the Euler flux extrapolation.
jdiss	1	A standard eigenvalue limiter is used in the flux extrapolation.
epsj	0.3	The magnitude of the eigenvalue limiter is 0.3 in the *j* direction
kflx	4	The Euler flux extrapolation method to use in the *k* direction is MUSCL Steger-Warming with Δp.
kord	3	The Euler flux extrapolation order of accuracy is third-order upwind biased.

Block #1 Flags (cont.)	Setting (cont.)	Explanation (cont.)
omgk	2.0d0	The value of ω to employ in the MUSCL scheme is 2.
klim	1	The Minmod flux limiter is used in the Euler flux extrapolation.
kdiss	0	No eigenvalue limiter is used in the flux extrapolation in the k direction.
epsk	0.03	This value is ignored because `kdiss=0`.
iextst	-1	The time advancement method used when simulating this master block will be implicit data parallel line relaxation.
nrlx	4	Four implicit data parallel line relaxation steps will be used in simulating this master block.
ildir	0	The lines will be formed automatically in an appropriate direction when simulating this master block.
ibcu	1	Implicit boundary conditions will be updated during each line relaxation step.
iblag	-1	Implicit boundary conditions will not be lagged when simulating this master block.
ilt	-1	Global timestepping will be employed when simulating this master block.
ibdir	1	This value is ignored because `nblk=2.`
cflm	1.0d20	This value is ignored because `ilt=-1` (not 2 or -2).
imin	20	Use a zonal interface boundary condition at this computational cell face.
imax	20	Use a zonal interface boundary condition at this computational cell face.
jmin	20	Use a zonal interface boundary condition at this computational cell face.
jmax	19	w=-w is the plane of symmetry at this computational cell face.
kmin	26	The wall at this computational cell face is set to catalytic radiative equilibrium.
kmax	1	The boundary conditions at this computational cell face are fixed at freestream conditions.

Block #2 Flags	Setting	Explanation
ntx	48	There are 48 computational cells in the *x* direction in Block #2.
nty	64	There are 64 computational cells in the *y* direction in Block #2.
ntz	64	There are 64 computational cells in the *z* direction in Block #2.
iconr	-1	This value is ignored because `iinit=-1`.
isim	1	This block will be included in the simulation.
ifree	1	Use freestream specification #1 for this master block.
initi	1	Use freestream specification #1 to initialize the interior of this master block.
ibadpt	1	Grid adaption will be performed on this block.
iflx	4	The Euler flux extrapolation method to use in the *i* direction is MUSCL Steger-Warming with Δp.
iord	3	The Euler flux extrapolation order of accuracy is third-order upwind biased.
omgi	2.0d0	The value of ω to employ in the MUSCL scheme is 2.
ilim	1	The MinMod flux limiter is used in the Euler flux extrapolation.
idiss	1	A standard eigenvalue limiter is used in the flux extrapolation.
epsi	0.3	The magnitude of the eigenvalue limiter is 0.3 in the flow direction
jflx	4	The Euler flux extrapolation method to use in the *j* direction is MUSCL Steger-Warming with Δp.
jord	3	The Euler flux extrapolation order of accuracy is third-order upwind biased.
omgj	2.0d0	The value of ω to employ in the MUSCL scheme is 2.
jlim	1	The MinMod flux limiter is used in the Euler flux extrapolation.
jdiss	1	A standard eigenvalue limiter is used in the flux extrapolation.
epsj	0.3	The magnitude of the eigenvalue limiter is 0.3 in the *j* direction
kflx	4	The Euler flux extrapolation method to use in the *k* direction is MUSCL Steger-Warming with Δp.
kord	3	The Euler flux extrapolation order of accuracy is third-order upwind biased.
omgk	2.0d0	The value of ω to employ in the MUSCL scheme is 2.

Block #2 Flags (cont)	Setting (cont)	Explanation (cont)
klim	1	The MinMod flux limiter is used in the Euler flux extrapolation.
kdiss	0	Flux extrapolation will not be performed in the *k* direction.
epsk	0.03	This value is ignored because `kdiss=0`.
iextst	-1	The time advancement method used when simulating this master block will be implicit data parallel line relaxation.
nrlx	4	Four implicit data parallel line relaxation steps will be used in simulating this master block.
ildir	0	The lines will be formed automatically in an appropriate direction when simulating this master block.
ibcu	1	Implicit boundary conditions will be updated during each line relaxation step.
iblag	-1	Implicit boundary conditions will not be lagged when simulating this master block.
ilt	1	Global timestepping will be employed when simulating this master block
ibdir	1	This value is ignored because `nblk=2.`
cflm	1.0d20	This value is ignored because `ilt=-1`
imin	20	Use a zonal interface boundary condition at this computational cell face.
imax	3	Use a first order extrapolation (supersonic exit) boundary condition at this computational face.
jmin	19	w=-w is the plane of symmetry at this computational cell face.
jmax	19	w=-w is the plane of symmetry at this computational cell face.
kmin	26	The wall at this computational cell face is set to catalytic radiative equilibrium.
kmax	1	The boundary conditions at this computational cell face are fixed at freestream conditions.

Freestream Specifications Flags	Setting	Explanation
irm	3	Velocity will be used as input for this area of the freestream.
density	1.6313d-5	The density of this area of the freestream is .000016313kg/m^3.
M/Re/V	3.1045d4	The velocity of this area of the freestream is 31,045 m/sec.
cx	0.8090160044	The cosine of the velocity vector in the *x* direction is 0.8090160044, i.e., cx=8 cos (34.2 degrees)
cy	0.5877852523	The cosine of the velocity vector in the *y* direction is 0.5877852523.
cz	0	The cosine of the velocity vector in the *z* direction is 0 because the flow vector is defined as 34.2 degrees in the *x-y* plane.
Tin	140.3	The translational temperature in this area of the freestream is 140.3 degrees Kelvin.
Trin	140.3	The rotational temperature in this area of the freestream is 140.3 degrees Kelvin.
Tvin	140.3	The vibrational temperature in this area of the freestream is 140.3 degrees Kelvin.
Tein	140.3	The free electron temperature in this area of the freestream is 140.3 degrees Kelvin.
Turbi	0.001d0	This value is ignored because `ivis`=1, defining the problem as a laminar flow simulation and telling DPLR to ignore turbulence- and transition-related flags.
Tkref	0.00d0	This value is ignored because `ivis`=1, defining the problem as a laminar flow simulation and telling DPLR to ignore turbulence- and transition-related flags.
subp0	2.650d2	This value is ignored because no subsonic boundary conditions are identified for this simulation.
subT0	2.650d2	This value is ignored because no subsonic boundary conditions are identified for this simulation.
pback	1.05d5	This value is ignored because no subsonic boundary conditions are identified for this simulation.
Cs H2	0.6822392	The fraction of the freestream mass contributed by H2 is 0.6822392.

Freestream Specifications Flags (cont.)	Setting (cont.)	Explanation (cont.)
cs H	0	The fraction of the freestream mass contributed by H is 0.
cs H+	0	The fraction of the freestream mass contributed by H+ is 0.
cs He	0.3177608	The fraction of the freestream mass contributed by He is 0.3177608
cs e	0	The fraction of the freestream mass contributed by electrons is 0.

CFL numbers or timesteps for ramping	Setting	Explanation
	.00001	Perform 20 iterations of the simulation at a CFL setting of .00001
	.0001	Perform 20 iterations of the simulation at a CFL setting of .0001
	.001	Perform 20 iterations of the simulation at a CFL setting of .001
	.01	Perform 20 iterations of the simulation at a CFL setting of .01
	.1	Perform 20 iterations of the simulation at a CFL setting of .1
	1	Perform 20 iterations of the simulation at a CFL setting of 1
	5	Perform 20 iterations of the simulation at a CFL setting of 5.
	10	Perform 20 iterations of the simulation at a CFL setting of 10.
	20	Perform 20 iterations of the simulation at a CFL setting of 20.
	50	Perform 20 iterations of the simulation at a CFL setting of 50.
	100	Perform 20 iterations of the simulation at a CFL setting of 100.
	200	Perform 20 iterations of the simulation at a CFL setting of 200.
	500	Perform 20 iterations of the simulation at a CFL setting of 500.
	750	Perform 20 iterations of the simulation at a CFL setting of 750.
	1000	Perform 20 iterations of the simulation at a CFL setting of 1000.
	2000	Perform 20 iterations of the simulation at a CFL setting of 2000.

CFL numbers or timesteps for ramping (cont.)	Setting (cont.)	Explanation (cont.)
	5000	Perform 20 iterations of the simulation at a CFL setting of 5000.
	-1	Stop reading CFL numbers.

4.3.3 Neptune Output Summary

To run this problem in DPLR, type a command at the prompt that is similar to:

mpirun –np 42 $DPLRBINDIR/dplr3d<neptune.inp

Upon execution, DPLR will create an on-screen summary, also known as a "standard out" of the problem as shown below.

```
*****************************************
 dplr3d
 NASA Ames Version 4.01.0
 Maintained by Mike Wright;  last modified: 02/05/09
 *****************************************

# Running on   8 processors
# --> Allocating   1 nodes to block   1
# --> Allocating   7 nodes to block   2
# --> Total load imbalance =   3.92%
# --> Input grid file hardwired for   8 processors

Executable Information
# --> built by twhite on Thurs Feb 5 17:23:43 PST 2009
# --> at host m100
# --> running Linux 2.6.9-42.0.2.ELsmp x86_64

Makefile Settings
# --> LD_LIBRARY_PATH = /opt/intel/fce/9.1.037/lib
/opt/ompi1.1.2/lib
# --> LFLAGS = /home/atipa/hpl/libgoto_opteron-64-r0.99-3.so
/home/atipa/hpl/xerbla.o /home/lib.working/tecio64.a
/usr/lib/gcc/x86_64-redhat-linux/3.4.5/libstdc++.a
# --> CPPFLAGS = -cpp -D_i686linuxipf
# --> FFLAGS = -r8 -extend_source -O3 -pad -ip -W0 -cm
# --> F77 = /opt/ompi1.1.2/bin/mpif90
# --> FXDRLIB = /home/lib/libfxdr.a
```

```
# Summary of enabled CPP compiler directives:
# --> AMBIPOLAR =  1
# --> PARKTEXP = 0.50
# --> NOHTC


****WARNING: CPP macro AMBIPOLAR = 1
             uses a simplistic model for ambipolar diffusion


# INFORM: Compiled for 32-bit compatible execution

# Overset Logic is disabled

# Dual time stepping is disabled

# Neptune Mechanism: 5 species, 5 reactions (Liebowitz 1973 & 1976)
Model
# --> Species List: H2 H H+ He e
# --> Reaction rates from: neptune5sp_leibowitz76.chem
# --> Reaction Status: 1 1 1 1 1
# --> Keq Fit Used   : 0 0 0 0 0
# --> Park 1990 fits for Keq (n=10^16)
# --> Assume molecules created/destroyed at mixture Tve

# Catalytic wall BC enabled
# --> Constant accomadation coeff;  gamma = 1.000
# --> Fully catalytic to ion recombination

# Radiative equilibrium BC enabled
# --> Constant wall emissivity;   epsilon = 0.85
# --> Maximum wall temperature = 3000.00 K

# Rotational Equilibrium - Fully Excited

# Vibrational Equilibrium - SHO

# Electronic Energy Neglected
# --> Assuming free electrons are coupled with T

# Laminar Navier-Stokes Simulation
# -->Gupta-Style Collision Integrals & Yos Mixing Rule
# -->Fickian Diffusion(Mass Fraction Gradients);Schmidt Number= 0.50
# -->SCEBD model used to compute diffusive fluxes

# Ideal Gas Equation of State

# 3-Dimensional Flow

# Implicit - Data Parallel Line Relaxation; nrlx =  4
# --> Using Global Timestepping

# Estimate   187MB stack memory required per PE

# Reading grid file: neptune-8PE.pgrx
```

```
# --> Reading block   1: grid cell size   32X  16X  64
# --> Reading block   2: grid cell size   48X  64X  64
# --> Total number of grid cells =   229376
# --> Computing grid dummy cells

# Freestream Reynolds    Number = 8.024E+04 (1/m)
# Freestream Frozen Mach Number = 3.715E+01
# Freestream Equil. Mach Number = 3.715E+01


nit =      1 rmsres =  1.0000000000000E+00  cfl = 1.0E-05
nit =      2 rmsres =  9.9999996847695E-01  cfl = 1.0E-05
nit =      3 rmsres =  9.9999993691276E-01  cfl = 1.0E-05
.
.
.
nit =     98 rmsres =  3.8595952079619E-01  cfl = 1.0E-01
nit =     99 rmsres =  3.8034985848052E-01  cfl = 1.0E-01
nit =    100 rmsres =  3.7508579577703E-01  cfl = 1.0E-01

# writing restart file: neptune.pslx
# solution written at: Thurs Feb  5 07:21:13 2009


nit =    101 rmsres =  3.7017658887905E-01  cfl = 1.0E+00
nit =    102 rmsres =  3.2830292631577E-01  cfl = 1.0E+00
nit =    103 rmsres =  3.0949864923525E-01  cfl = 1.0E+00
.
.
.
nit =    198 rmsres =  1.5724959884754E-02  cfl = 5.0E+01
nit =    199 rmsres =  1.5236815979400E-02  cfl = 5.0E+01
nit =    200 rmsres =  1.4743944713869E-02  cfl = 5.0E+01

# writing restart file: neptune.pslx
# solution written at: Thurs Feb  5 07:35:27 2009


nit =    201 rmsres =  1.4303718730322E-02  cfl = 1.0E+02
nit =    202 rmsres =  1.7744774664322E-02  cfl = 1.0E+02
nit =    203 rmsres =  1.7218162636161E-02  cfl = 1.0E+02
.
.
.
nit =    298 rmsres =  2.3267700178866E-05  cfl = 1.0E+03
nit =    299 rmsres =  2.1545250679943E-05  cfl = 1.0E+03
nit =    300 rmsres =  2.0022263684723E-05  cfl = 1.0E+03

# writing restart file: neptune.pslx
# solution written at: Thurs Feb  5 07:49:42 2009


nit =    301 rmsres =  1.8676626863417E-05  cfl = 2.0E+03
nit =    302 rmsres =  1.9613965114431E-05  cfl = 2.0E+03
nit =    303 rmsres =  2.1038915489521E-05  cfl = 2.0E+03,
```

```
.
.
.
nit =    398 rmsres =  1.4656581391990E-08  cfl = 5.0E+03
nit =    399 rmsres =  1.3612594864547E-08  cfl = 5.0E+03
nit =    400 rmsres =  1.2656218641573E-08  cfl = 5.0E+03

# writing restart file: neptune.pslx
# solution written at: Thurs Feb  5 08:03:56 2009


nit =    401 rmsres =  1.1780771213662E-08  cfl = 5.0E+03
nit =    402 rmsres =  1.0979147316707E-08  cfl = 5.0E+03
nit =    403 rmsres =  1.0245683275129E-08  cfl = 5.0E+03
.
.
.
nit =    498 rmsres =  2.1745095792356E-10  cfl = 5.0E+03
nit =    499 rmsres =  2.0982611245646E-10  cfl = 5.0E+03
nit =    500 rmsres =  2.0246851605138E-10  cfl = 5.0E+03

# writing restart file: neptune.pslx
# solution written at: Thurs Feb  5 08:18:10 2009



# Loop time =    4227.63 seconds on   8 processors
```

Figure 4-4 Standard Out for DPLR Run of Neptune Probe

4.3.4 Neptune Output Summary Information

In addition to verifying the values entered into the DPLR input deck, the DPLR output summary displays information about computing resources required for the run, values calculated by code, and an initial snapshot of the set of iterations that are being performed as the problem converges to a solution.

In this sample case, DPLR estimates that 187 megabytes of stack memory will be required for each of the 8 processors, calculates the Reynolds and Mach numbers used in the simulation, shows that the early iterations of the run began with very large residuals and and very small CFL timesteps, writes restart files every 100 iterations, and ends the run with a very small residual at iteration 500 when the neptune.pslx solution file is written. (Note that this estimate may not include additional memory requirements for turbulence models.)

4.4 Monitoring the DPLR Run

When DPLR begins its execution of a simulation run, the screen will display the standard out as discussed above in Section 4.3.4.

In addition to the standard out, you can actively monitor the simulation run by setting up a POSTFLOW input deck to read restart files as they are being saved during the DPLR run. As the simulation progresses, you can extract the data you want to examine and launch Tecplot (or some other graphics visualization program) to read the POSTFLOW output files and create a graphic representation of the state of your solution at specific iterations or timesteps. (See Chapter 5 for more information on Using POSTFLOW)

This workflow set-up can help you monitor the progress of your simulation run early enough to see if you are accurately capturing flow conditions along the shock wave. If not, you may be able to use a runtime control file to implement a grid adaption process during the run to improve the quality of the simulation. (See Section 6.4 for more information on runtime control files).

Ideally, your simulation will achieve convergence when the residual from the latest iteration in your solution approaches zero and the resulting data visualization accurately represents the flow conditions you are simulating as shown in the standard out for the Neptune simulation at the 500^{th} iteration. However, each case will have its own set of unique convergence parameters to tell you when you have achieved an acceptable solution. In practice, if your solution progresses far enough for the residual to stop dropping by orders of magnitude over time and appears to level out, you may have achieved an acceptable result.

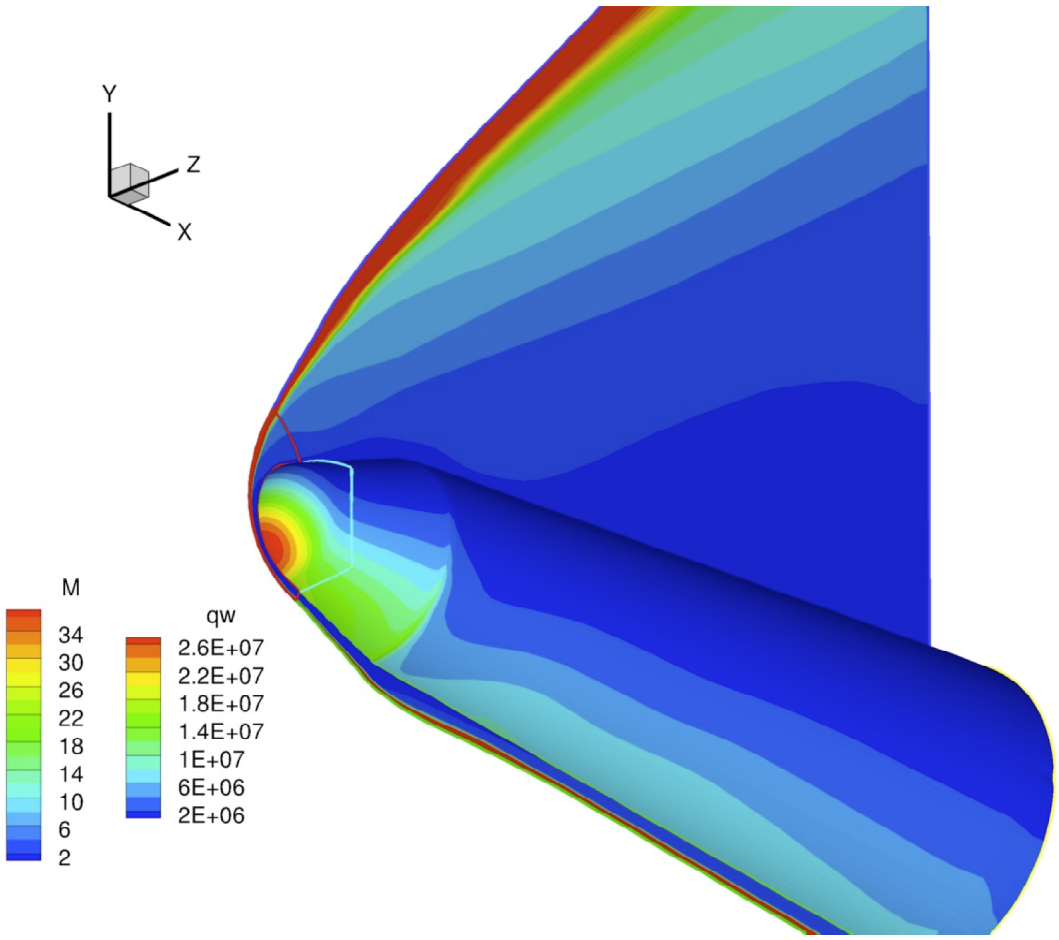


Figure 4-5 Graphic Representation of Mach Contours and Convective Heating at the Wall in the Neptune Simulation after 500 Iterations.

Chapter 5 - Using POSTFLOW

Contents

5.0 Introduction

Restart files generated by DPLR contain all the input deck settings and physical modeling parameters that were used in the simulation. These data exist independently of the original input files used to run the simulation.

Using POSTFLOW, you can identify and extract specific data from a DPLR restart file to use in a presentation or further process with graphics software (such as Tecplot) to create appropriate visualizations of the results of your CFD simulation.

POSTFLOW always runs in serial mode on a single processor, regardless of the number of processors used to run the simulation that generated the restart file.

Because DPLR maintains backward compatibility, the current version of POSTFLOW can be used to post-process restart files generated with earlier versions of the DPLR Code Package.

5.1 Running POSTFLOW

Step 1: **Open** the text editor program for your system.

Action: At the command line prompt, type:

/[path to your post directory]/`post_flow3d_mb.inp`

Result: An input file appears on screen with placeholder default values. To start with a blank deck, delete the default values as shown on the following page.

```
Input file for postflow


imemmode itruev  ifstat

inrest   ingrid  inbcf   ouform  iwrtd

interp   nzones   isep     istyp   iunits

lref   aref   xmc   ymc   zmc   imrx  imry  imrz

iwind   cxs   cys   czs

iexbc    <== list of BC numbers to extract from dataset

ivarp  <== list of variable numbers to extract from dataset

Tecplot/plot3d zone information:
iwrt ifac imin imax jmin jmax kmin kmax  bkmin  bkmax  zonetitle
                                                       'stag2d'
                                                       'body2d'
                                                       'flow2d'
                                                       'terminator'

fname,pname,(gname),(bname)
```

Figure 5-1 POSTFLOW Input Deck

Step 2: **Enter** problem-specific values for each of the input variables or "flags". (See Section 5.2 for a description of input flags and a list of allowable values.)

Step 3: **Rename** and **Save** your POSTFLOW input file to your working directory.

Step 4: **Run** POSTFLOW.

Action: At the command line prompt, type:

```
postflow < yourpostflowfilename.inp
```

Result: An output file that can be processed by a third-party graphics program (such as Tecplot) is created according to your specifications along with an on-screen summary of

actions performed by POSTFLOW. (See Section 5.3 for an example of a problem-specific POSTFLOW input deck and the output summary.)

5.2 Input Flags for POSTFLOW

Input variables for POSTFLOW are discussed below in the order they appear in the deck.

`imemmode` – Specifies the memory mode selected for running POSTFLOW. Allowable values are:

1 low memory mode

2 high memory mode (*Recommended*)

***Tech Tip**: Using high memory mode makes all the features of POSTFLOW available and is the recommended setting. If your computing resources are insufficient for running in this mode, i.e., capable of holding all flow variables for the largest physical block in the simulation at any one point in time, then choosing the low memory mode will enable you to use the program, but will require significantly longer processing times.*

`itruev` – Specifies the method to use to compute derivative values, such as skin friction or heat transfer. Allowable values are:

0 evaluate derivatives using a 1st-order approximation

1 evaluate derivatives using accurate 2nd-order expressions *(Recommended)*

***Tech Tip**: When `immode=1` (low memory), extraction of true derivatives is not possible. In this case, POSTFLOW automatically sets `itruev=0`, then echoes a warning to the screen.*

`ifstat` – Specifies the type of statistical processing POSTFLOW performs on flow variables (if available). Allowable values are:

0 process instantaneous flow variables (default)

1 compute mean (exact for primitive quantities only)

2 compute root mean square (RMS) (exact for primitive quantities only)

3 compute standard deviation *(not available in DPLR 4.01.1)*

inrest – Specifies the format of the restart file to be read by POSTFLOW. Allowable values are:

1 parallel archival file (native unformatted)

11 parallel archival file (XDR format)

21 Parallel archival file (ASCII)

ingrid – Specifies the format of the grid file to be used when post-processing the simulation data. Allowable values are:

0 get format from restart file *(Recommended)*

1 parallel archival file (native unformatted)

11 parallel archival file (XDR format)

21 parallel archival file (ASCII)

***Tech Tip**: Setting `igrid=0` ensures that the grid file being used to post-process the data is the same as the one used to generate the data in the first place. However, if the name of the grid file, or its location relative to the restart file is ever changed, you must use one of the other settings in `igrid` to point POSTFLOW to the original grid file,*

---

inbcf – Specifies the format of the boundary condition file, if any, that was used to generate the data in the restart file being read by POSTFLOW. Allowable values are:

0 get format from restart file *(Recommended)*

1 parallel archival file (native unformatted)

11 parallel archival file (XDR format)

21 parallel archival file (ASCII)

***Tech Tip**: POSTFLOW will automatically look at the restart file to determine if any boundary condition file is required and what the format is. If no BC file was used during the simulation, the value of `inbcf` is ignored.*

`ouform` – Specifies the desired format of the output data. Allowable values are:

2	plot3d grid or q-file (native unformatted)
3	plot3d grid or function file (native unformatted)
5	Tecplot block ordered data binary
6	Tecplot point ordered data binary
7	compute max/min values for variables and output to STDOUT
8	integrate variables over given surface(s) and output to STDOUT
9	RESERVED
10	print selected freestream quantities to STDOUT
11	output datasets for ***Moment*** calculations
17	compute max/min & maxloc/minloc and output to STDOUT
18	print a list of NaN locations to STDOUT
22	plot3d grid or q-file (ASCII)
23	plot3d grid or function file (ASCII)
25	Tecplot block ordered data ASCII
26	Tecplot point ordered data ASCII
28	RADEQUIL LOS file (ASCII)
32	gzipped plot3d grid or q-file (ASCII)
33	gzipped plot3d grid or function file (ASCII)
110	print freestream quantities to STDOUT in tabular format

Tech Tips:
1). Formats for the output files will usually be `plot3d` *(*`ouform=3,23,33`*) - standard CFD output formats that can be read by most commercial post-processing tools - and Tecplot (*`ouform=5,6,25,26`*) – a file format used only by Amtec's Tecplot post processing visualization software .*

2) The `plot3d q` *file output option (*`ouform=2`*) is included primarily for historical purposes. Although still technically available, writing data to a* `q` *file requires the user to know and include the specific variable set specified by that file format. With no error checking performed by DPLR, data file validity becomes the*

entire responsibility of the user.

3). To generate Tecplot binary files (`ouform=5,6`), the Amtec-provided "tecio.a" (or "tecio64.a") runtime library must be installed on your system.

`iwrtd` – Specifies whether a subdirectory called `INPUTDECKS` containing reconstructions of the DPLR input decks (including the physical property data decks) used to run the simulation will be created in your working directory. Allowable values are:

- 0 do not create a subdirectory containing reconstructed input decks
- 1 create a subdirectory containing reconstruct input decks *(recommended)*

***Tech Tip**: One of the more powerful features of POSTFLOW is the ability to recreate usable DPLR input and physical data decks directly from the restart file. Because of this, it is always possible to determine the settings and physical constants used to generate the simulation, even if the original DPLR input deck has been altered or misplaced.* (*Note*: *Although POSTFLOW can process restart files generated by previous versions of DPLR, DPLR input decks reconstructed and saved in the subdirectory created by POSTFLOW will always be generated in the format of the current version of the DPLR Code Package.)*

`interp` – Specifies how cell-centered finite-volume flow data are represented on a node-centered grid. Allowable values are:

- 0 move flow data to the lower-left cell *(least accurate)*
- 1 interpolate grid points to cell centers *(See Tech Tip #1)*
- 2 interpolate flow data to grid points *(See Tech Tip #2)*
- 11 interpolate grid points to cell centers; no boundary points *(See Tech Tip #3)*
- 21 interpolate grid points to cell centers; even at boundaries *(See Tech Tip #4)*

***Tech Tips**:*
***1)**. Generates cell-centered grids adding additional face-centered points to the boundaries. Flow quantities are not interpolated to the interior of the grid, thereby holding distortion of output data to a minimum. Because output grid points lie at the cell centers of the original CFD grid, the output grid resulting from* `interp=1` *cannot be used to run further CFD simulations.*

***2)** Preserves the location of the CFD grid points and interpolates finite-volume data onto these mesh points. Best to use when output data is to be processed using **SAGe** or a utility such as **Outbound** to move the outer boundary of the grid or adapt the grid to the computed flowfield.*

***3)** Identical to* `interp=1` *except that additional points are not added at the block boundaries, so the output grid will have "holes" in it along those boundaries. Best to use for computing integrated forces and moments, or for outputting pointwise forces for later offline integration using the **Moment** utility program. See Section 9.1.5 for more information on **Moment**.*

***4)**. Identical to* `interp=1` *except that even the points at the boundaries are located using cell-centered interpolation. Maximum output dimensions using this option is the number of cells in each computational direction, plus two points in each direction representing the points added within the boundaries. Primarily used for debugging by code developers to gain access to the cell-centered values of quantities in the grid dummy cells rather than the face-centered values available using* `interp=1`.

`nzones` – Specifies the maximum number of output data zones to be generated.

Recommended value = 20.

***Tech Tip**: Used by POSTFLOW to size certain output arrays, a moderate value such as the recommended 20 should be sufficient. However, if this value is too small for the output arrays you ask POSTFLOW to generate, the program will abort and generate an error message prompting you to increase the* `nzones` *value.*

isep – Specifies whether multiple output datasets are to be written to a single or multiple files Allowable values are:

0 all active output datasets are written to a single file

1 each active output dataset is written to its own file

istyp – Specifies how to extract boundary condition data with iexbc. Allowable values are: (*Not working in DPLR 4.01.1)*

-1 extract entire volume of data for each boundary condition

1 extract a single plane of data for each boundary condition

-2 extract 2 planes of data for each boundary condition

iunits – Specifies whether a POSTFLOW will include SI units associated with data in output files formatted for Tecplot (`ouform=5:6, 25:26`). Allowable values are:

0 do not include units in the output file

1 include SI units in the output file

***Tech Tip**: If `outform` is set to create output files NOT specifically formatted for Tecplot, this flag will be ignored*

lref – Specifies the reference length, in SI units (meters), used for the normalization of moment coefficients.

***Tech Tip**: The extraction of moments and moment coefficients can either be performed directly in POSTFLOW or with an included utility program **Moment**. If **Moment** is used, the value you enter into 1ref will be passed to the ultility for use in computation.*

aref – Specifies the reference area, in SI units (square meters), used for the normalization of force and moment coefficients.

xmc, ymc, zmc – Indicates the *x, y, z* position, in meters, as specified in the input `plot3d` grid file of the moment reference center used for extracting moments and moment coefficients.

***Tech Tip**: Although extraction of hinge moments for control surfaces is not currently supported in POSTFLOW, it can be accomplished using the **Moment** utility.*

`imrx,imry,imrz-` Specifies planes of symmetry used in the simulation. Allowable values are:

0 do not enforce symmetry about this plane

1 enforce symmetry about this plane

***Tech Tips**:*

1). Possible planes of symmetry are defined as:

imrx – body is symmetric about the yz-plane
imry – body is symmetric about the xz plane
imrz – body is symmetric about the xy plane

2) POSTFLOW currently supports bilateral symmetry (any one of the above flags =1) and quadrilateral symmetry (any two of the above flags =1) .

3) If the symmetry of the vehicle is more complex than a simple bilateral or quadrilateral representation, set all of the above flags=0 and compute the symmetry relations off-line after post-processing is complete.

4) If ouform=8 requesting integrated variable reporting over given surface(s) and ivarp=600:673; 700:773 requesting force or moment coefficients, it is important to set `aref` to specify the full reference area when normalizing these computed forces if the symmetry flags are also used.

5) All three symmetry flags are valid for 3D flows, and none are valid for a 2D or axisymmetric flow.

`iwind -` Specifies the velocity vector orientation axis (aka global "wind") for calculation of certain output variables (i.e., ivarp values). Allowable values are:

0 do not alter the raw output data

1 determine sign by a dot product with freestream vector *(recommended for simulations with only one freestream specification)*

2 determine sign by a dot product with supplied wind vector *(recommended for simulations with more than one freestream specification)*

***Tech Tip**: Because the global wind axis is used either to determine the sign of the output skin friction (shear stress) or to convert output forces into a wind-oriented (lift and drag) coordinate system,* `iwind` *will be ignored unless* `ivarp`*=600:673; 700:773.*

`cxs, cys, czs -` Specifies the direction cosines of the global wind axis in the *xyz* directions when `iwind=2`.

***Tech Tips**:*

1) These are defined as unit metrics, such that:

$$cxs^2 + cys^2 + czs^2 = 1$$

and the components u, v, and w of the freestream velocity vector V are given by $u = V \bullet cxs$; $v = V \bullet cxy$; $w = V \bullet cxz$

2) User input values are always normalized by POSTFLOW to ensure that these expressions are valid.

`iexbc -` Boundary condition number(s) for the wall or surface from which data will be extracted. Allowable values, listed in Section 4.2 for the DPLR input flag `ibc`, must be an array of comma or space separated entries. Entering a value of -1 disables this feature.

***Tech Tips**:*

1) To extract data from the intersection of two surfaces, enter a reference boundary value, then a forward slash, then the boundary condition that is desired to intersect with the reference boundary. This expression becomes one value and can then be added to the array of numbers on this input line. For example:

```
iexbc
26/18 3
```

would extract data from the intersection between a catalytic radiative equilibrium wall (`ibc=26`) and the y symmetry plane

(ibc=18) in addition to extracting data from the supersonic exit plane (ibc=3).(Note that the order of the two numbers is important as the wall must be defined before an intersecting plane of symmetry can be specified.)

2) If the simulation contains multiple instances of a boundary condition specified in **iexbc**, *the resulting data extraction will be saved as separate "blocks" for a plot3d output file or "zones" for a Tecplot output file. Both designations refer to the same regions in the simulation and will be named by the* **iexbc** *setting. For example, if iexbc=19, POSTFLOW will display the words zone t=BC19 for each block in which that boundary condition is extracted.*

3) Entering appropriate values into **iexbc** *is a quick and easy way to extract defined surface data from a complex multiblock grid and can be used with, or instead of, zone specification extraction as defined below.*

---

ivarp – Specifies the flow variables to be extracted from the restart file. Entries must be an array of comma- or space-separated integers. Allowable values are:

Grid Coordinates

0 *all grid coordinates*

1 *x*-coordinate (x)

2 *y*-coordinate (y)

3 *z*-coordinate (z)

Grid-Related Variables

10 *all path-lengths*

11 path length along grid lines in *i*-direction (si)

12 path length along grid lines in *j*-direction (sj)

13 path length along grid lines in *k*-direction (sk)

14 *unit outward normal *x*-direction cosine (sx)

15 *unit outward normal *y*-direction cosine (sy)

16 *unit outward normal *z*-direction cosine (sz)

21 *body normal distance (dn)

22 *deviation from orthogonality [deg.] (dev)

23 *face area (Area)

25 cell aspect ratio (dmax/dmin) (CAR)

Mixture Transport Properties

50 total viscosity (mu)
51 total kinematic viscosity (nu)
52 total translational thermal conductivity (kap)
53 total rotational thermal conductivity (kapr)
54 total vibrational thermal conductivity (kapv)
55 free electron thermal conductivity (kape)
56 total binary diffusion coefficient (D)
57 mixture mean free path (mfp)
58 unit Reynolds number (Re/L)
59 cell Reynolds number (Re_c)

Thermodynamic Properties

60 ratio of frozen specific heats cp/cv (G)
61 frozen specific heat at constant volume (cv)
62 frozen specific heat at constant pressure (cp)
63 translational specific heat at constant volume (cvt)
64 rotational specific heat at constant volume (cvr)
65 vibrational specific heat at constant volume (cvv)
66 electronic specific heat at constant volume (cve)
68 mixture gas constant (R)
69 mixture molecular weight (Mw)

Turbulence Quantities

70 turbulent kinetic energy (TKE)
71 turbulent omega (omega_t)
72 RESERVED
73 RESERVED
75 Spalart-Almaras conserved variable (mu_SA)

Laminar Transport Properties

80 laminar viscosity (mu_l)

81 laminar kinematic viscosity (nu_l)
82 laminar thermal conductivity (kap_l)
83 laminar rotational thermal conductivity (kapr_l)
84 laminar vibrational thermal conductivity (kapv_l)
85 laminar free electron thermal conductivity (kape_l)
86 laminar binary diffusion coefficient (D_l)
87 laminar Lewis number (Le)
88 laminar Schmidt number (Sc)
89 laminar Prandtl number (Pr)

Turbulent Transport Properties

90 turbulent eddy viscosity (mu_t)
91 turbulent kinematic eddy viscosity (nu_t)
92 turbulent thermal conductivity (kap_t)
93 turbulent rotational thermal conductivity (kapr_t)
94 turbulent vibrational thermal conductivity (kapv_t)
95 turbulent free electron thermal conductivity (kape_t)
96 turbulent binary diffusion coefficient (D_t)
97 turbulent Lewis number (Le_t)
98 turbulent Schmidt number (Sc_t)
99 turbulent Prandtl number (Pr_t)

Mixture Flow Properties

(Note that stagnation quantities (density, pressure, and temperature) are computed assuming isentropic relations, and thus are not valid for a chemically reacting flowfield.)

100 mixture density (rho)
101 mixture number density (N_tot)
102 stagnation mixture density (r_o)
110 pressure (p)
111 dynamic pressure (Q)
112 stagnation pressure (p_o)
113 Pitot pressure (p_pitot)
114 pressure coefficient (C_p)

120 translational temperature (T)
121 bulk temperature (T_b)
122 stagnation temperature (T_o)
124 rotational temperature (Tr)
125 vibrational temperature (Tv)
126 electronic temperature (Te)
127 free electron temperature (Tel)
132 total enthalpy per unit mass (h)
133 static enthalpy per unit mass (h_s)
134 total enthalpy per unit volume (rh)
135 static enthalpy per unit volume (rh_s)
142 total energy per unit mass (e)
143 total translational energy per unit mass (et)
144 total rotational energy per unit mass (er)
145 total vibrational energy per unit mass (ev)
146 total electronic energy per unit mass (ee)
147 total free electron energy per unit mass (eel)
148 total chemical formation energy per unit mass (eh)
149 total kinetic energy per unit mass (eU)
150 velocity in the *x*-direction (u)
151 velocity in the *y*-direction (v)
152 velocity in the *z*-direction (w)
153 velocity magnitude (Vel)
154 frozen Mach number (M)
155 frozen speed of sound (a)
156 mean thermal speed (cbar)
157 normalized velocity in the *x*-direction (u/Vel)
158 normalized velocity in the *y*-direction (v/Vel)
159 normalized velocity in the *z*-direction (w/Vel)
160 momentum per unit volume in the *x*-direction (rhou)
161 momentum per unit volume in the *y*-direction (rhov)
162 momentum per unit volume in the *z*-direction (rhow)
163 total energy per unit volume (re)
164 total rotational energy per unit volume (rer)

165 total vibrational energy per unit volume (rev)
166 total electronic energy per unit volume (ree)
167 total free electron energy per unit volume (rel)
168 total chemical formation energy per unit volume (reh)
169 total kinetic energy per unit volume (reU)
170 entropy (S)
175 pointwise unit radiative emission (Erad)
180 degree of ionization (zeta)
181 debye length (lam_D)
182 Tstar (Tstar)
183 electron charge (ec)
184 plasma frequency (wpe)
185 critical transmission frequency (wpecrit)
194 total energy per unit mass in rotational Eqn. (er_B)
195 total energy per unit mass in vibrational Eqn. (ev_B)
196 total energy per unit mass in electronic Eqn. (ee_B)
197 total energy per unit mass in free electron Eqn. (el_B)
202 *delta velocity at wall (Del_V)
204 *delta temperature at wall (Del_T)
250 velocity in the x-direction normalized by V_∞ (u/Vin)
251 velocity in the y-direction normalized by V_∞ (v/Vin)
252 velocity in the z-direction normalized by V_∞ (w/Vin)
324 limited rotational temperature (Tr_l)
325 limited vibrational temperature (Tv_l)
326 limited electronic temperature (Te_l)
327 limited free electron temperature (Tel_l)

Viscous Derivative-Based Quantities

501 *skin friction coefficient (Cf)
502 *unit viscous force on a face in *x*-direction (tau_x)
503 *unit viscous force on a face in *y*-direction (tau_y)
504 *unit viscous force on a face in *z*-direction (tau_z)
507 *total wall shear stress (tau)
511 *Stanton number [based on wall enthalpy] (Ch)

512 *Heat transfer coefficient in mass flux units (Chm)
517 *Stanton number [based on freestream conditions] (St)
518 *Convective heating coefficient (Ct)
520 radiative equilibrium heat transfer (Qeq)
521 *total wall heat transfer (qw)
522 *translational wall heat transfer (qT)
523 *rotational wall heat transfer (qR)
524 *vibrational wall heat transfer (qV)
525 *free electron wall heat transfer (qEl)
526 *catalytic wall heat transfer (qD)
527 *velocity wall heat transfer (qU)
531 *total wall heating (qwi)
581 *spacing in wall units y^+ (yp)
584 *tangential velocity in wall units u+ (up)
585 *normal velocity in wall unites v+ (vp)
591 *blowing velocity through face (vb)
594 *mass flow rate through face (mdot)
595 *unit mass flow rate through face (mdotU)
596 *thrust through face (Thrust)

Aerodynamic Forces and Moments

Force and moment variables (`ivarp=600:673, 700:773`) are usually extracted in conjunction with surface integration in order to generate integrated aerodynamic data and/or coefficients. Because an accurate surface integration cannot be performed if the data are extrapolated to zone edges, POSTFLOW will automatically set `interp` *=11 whenever one or more force and moment variables are specified as output. This will result in a surface mesh with gaps along all block boundaries if pointwise surface data are requested unless you perform an off-line integration using a utility program such as* ***Moment****.*

600 **total force on a face in all directions*
601 *total force on a face in *x*-direction (Fx)
602 *total force on a face in *y*-direction (Fy)
603 *total force on a face in *z*-direction (Fz)
604 *total force on a face in *x*-direction per unit area (Fx_a)

605 *total force on a face in *y*-direction per unit area (Fy_a)

606 *total force on a face in *z*-direction per unit area (Fz_a)

610 **pressure force on a face in all directions*

611 *pressure force on a face in *x*-direction (Fx_P)

612 *pressure force on a face in *y*-direction (Fy_P)

613 *pressure force on a face in *z*-direction (Fz_P)

614 *pressure force on a face in *x*-direction per unit area (Fx_Pa)

615 *pressure force on a face in *y*-direction per unit area (Fy_Pa)

616 *pressure force on a face in *z*-direction per unit area (Fz_Pa)

620 **viscous force on a face in all directions*

621 *viscous force on a face in *x*-direction (Fx_V)

622 *viscous force on a face in *y*-direction (Fy_V)

623 *viscous force on a face in *z*-direction (Fz_V)

624 *viscous force on a face in *x*-direction per unit area (Fx_Va)

625 *viscous force on a face in *y*-direction per unit area (Fy_Va)

626 *viscous force on a face in *z*-direction per unit area (Fz_Va)

650 **total force coefficient on a face in all directions*

651 *total force coefficient on a face in *x*-direction (Cx)

652 *total force coefficient on a face in *y*-direction (Cy)

653 *total force coefficient on a face in *z*-direction (Cz)

660 **pressure force coefficient on a face in all direction*

661 *pressure force coefficient on a face in *x*-direction (Cx_P)

662 *pressure force coefficient on a face in *y*-direction (Cy_P)

663 *pressure force coefficient on a face in *z*-direction (Cz_P)

670 **viscous force coefficient on a face in all direction*

671 *viscous force coefficient on a face in *x*-direction (Cx_V)

672 *viscous force coefficient on a face in *y*-direction (Cy_V)

673 *viscous force coefficient on a face in *z*-direction (Cz_V)

700 **total moment on a face in all directions*

701 *total moment on a face in *x*-direction (Mx)

702 *total moment on a face in *y*-direction (My)

703 *total moment on a face in *z*-direction (Mz)

710 **pressure moment on a face in all directions*

711 *pressure moment on a face in *x*-direction (Mx_P)

712 *pressure moment on a face in *y*-direction (My_P)

713 *pressure moment on a face in *z*-direction (Mz_P)

720 **viscous moment on a face in all directions*

721 *viscous moment on a face in *x*-direction (Mx_V)

722 *viscous moment on a face in *y*-direction (My_V)

723 *viscous moment on a face in *z*-direction (Mz_V)

750 **total moment coefficient on a face in all directions*

751 *total moment coefficient on a face in *x*-direction (Cmx)

752 *total moment coefficient on a face in *y*-direction (Cmy)

753 *total moment coefficient on a face in *z*-direction (Cmz)

760 **pressure moment coefficient on a face in all directions*

761 *pressure moment coefficient on a face in *x*-direction (Cmx_P)

762 *pressure moment coefficient on a face in *y*-direction (Cmy_P)

763 *pressure moment coefficient on a face in *z*-direction (Cmz_P)

770 **viscous moment coefficient on a face in all directions*

771 *viscous moment coefficient on a face in *x*-direction (Cmx_V)

772 *viscous moment coefficient on a face in *y*-direction (Cmy_V)

773 *viscous moment coefficient on a face in *z*-direction (Cmz_V)

Debugging/Status Information

980 pointwise icatmd numbers along block edges (icatmd)
981 pointwise ireqmd numbers along block edges (ireqmd)
990 pointwise BC numbers along block edges (ibcp)
991 net charge [should always be zero] (Qnet)
992 sum of mass fractions [should always be one] (Csum)
998 zero (zero)
999 pointwise L2Norm residual (res)

Species Data

The following variables are species-specific data. In each case the user can choose to extract data for either a subset of the species by entering just the desired variable numbers, or data for all species by entering the appropriate" macro" value. (See Tech Tip #2)

1000 *all species densities*
1000+*n* density of species *n* (n)

1200 *all species number densities*
1200+*n* number density of species *n* (N_n)

1400 *all species mass fractions*
1400+*n* mass fraction of species *n* (C_n)

1600 *all species mole fractions*
1600+*n* mole fraction of species *n* (X_n)

1800 *all species densities, normalized by* ρ_∞
1800+*n* normalized density of species n (RnD_n)

3400 *all species rotational temperatures*
3400+*n* rotational temperature of species *n* (Tr_n)

3600 *all species vibrational temperatures*
3600+*n* vibrational temperature of species *n* (Tv_n)

4000	*all species total internal energies per unit mass*
4000+*n*	total internal energy per unit mass of species *n* (e_n)
4200	*all species translational internal energies per unit mass*
4200+*n*	trans. internal energy per unit mass of species *n* (et_n)
4400	*all species rotational internal energies per unit mass*
4400+*n*	rotational internal energy per unit mass of species *n* (er_n)
4600	*all species vibrational energies per unit mass*
4600+*n*	vibrational energy per unit mass of species *n* (ev_n)
4800	*all species electronic energies per unit mass*
4800+*n*	electronic internal energy per unit mass of species *n* (ee_n)
5000	**all species mass flow rates through surface*
5000+*n*	*mass flow rate through surface of species *n* (mdot_n)
5200	**all species mass flow rates through surface [per unit area]*
5200+*n*	*mass flow rate through surface of species *n* (mdotU_n)
6000	*all species total specific heats at constant volume*
6000+*n*	total specific heat at constant volume of species *n* (cvx_n)
6200	*all species translational specific heats at constant volume*
6200+*n*	translational specific heat at const. vol. of species *n* (cvt_n)

6400	*all species rotational specific heats at constant volume*
6400+*n*	rotational specific heat at const. vol. of species *n* (cvr_n)
6600	*all species vibrational specific heats at constant volume*
6600+*n*	vibrational specific heat at const. vol. of species *n* (cvv_n)
6800	*all species electronic specific heats at constant volume*
6800+*n*	electronic specific heat at const. vol. of species *n* (cve_n)
7000	*all species frozen specific heats at constant pressure*
7000+*n*	specific heat at constant pressure of species *n* (cp_n)
7200	*all species frozen specific heats at constant volume*
7200+*n*	specific heat at constant volume of species *n* (cv_n)
8000	*all species gas constants*
8000+*n*	gas constant of species *n* (R_n)
8200	*all species equivalent degrees of freedom [nkT]*
8200+*n*	equivalent degrees of freedom of species *n* (dof_n)
8400	*all species partial pressures*
8400+*n*	partial pressure of species *n* (p_n)
8600	*all species mean thermal speeds*
8600+*n*	mean thermal speed of species *n* (cbar_n)
8800	*all species chemical formation energies per unit mass*
8800+*n*	formation energy per unit mass of species *n* (eh_n)

10000	*all species diffusion coefficients*
10000+*n*	diffusion coefficient of species *n* (D_n)
10200	*all species ambipolar diffusion effectiveness*
10200+*n*	ambipolar diffusion effectiveness of species *n* (DaC_n)
10400	*all species effective Schmidt numbers*
10400+*n*	effective Schmidt number of species *n* (Sc_n)
10800	*all species unit diffusion mass fluxes*
10800+*n*	unit diffusion mass flux of species *n* (MD_n)

***Tech Tips**:*
1) A single set of output variables may be specified for a given run of POSTFLOW. If a variable that is not permitted by the simulation specifications is selected for extraction, such as the coefficient of viscosity from an Euler simulation, POSTFLOW will remove it from the `ivarp` *array and echo a message to the screen.*

2) The list of species-specific variables includes some italicized "macro" selections that allow extraction of several related items. For example, `ivarp=1000` *tells POSTFLOW to output species densities for all species in the simulation, relieving you of the need to identify each species by its order number in the .chem file. Whenever macro values are used, only those variable relevant to the simulation will be extracted, so ivarp=0 will automatically extract x,y, and z coordinates for a 3D flow, but only x and y for a 2D or axisymmetric flow.*

3, Variables prefaced with an asterisk (*) are defined as surface-specific quantities and are extracted with respect to a given surface direction as defined either with the `ifac` *flag in the zone specifications (see below) or automatically determined when extracting surfaces with the* `iexbc` *flag.*

4). All extracted variables are output in SI units. Units for dimensional output variables are echoed to the screen when one of the standard output formats are specified.

Tecplot/Plot3D Zone Specification Flags – The flags in this section of the POSTFLOW input deck define the extent of data extraction required for specified flow volumes or "zones". In general, one row of data defines each desired extraction. The last line in this group is the "terminator" line in which `iwtr=-1` instructs the code to stop reading zone specification information. A terminator line must be present or a run time error will occur.

`iwrt` – Specifies whether or not POSTFLOW will perform data extractions for that line of the zone specification array. Allowable values are:

0	do not extract the data defined by this zone specification
1	extract the data defined by this zone specification
-1	terminator line

***Tech Tip**: You can enter any number of zone specification lines in the POSTFLOW input deck. However, only those that are turned on by `iwrt=1` will actually be extracted at runtime. This way, you can set up a default input deck with multiple zone specification lines for all possible desired output. Then, each time POSTFLOW is run, only the data that are actually required can be "turned on" while the rest are left inactive.*

`ifac` – Specifies the *ijk* orientation of the surface being extracted. Allowable values are:

0	No face selected
1	*i*-face
2	*j*-face
3	*k*-face

***Tech Tips**:*
1) `ifac` is only needed when surface-oriented variables are specified in `ivarp` (i.e., those marked with an asterisk, such as skin friction or heat transfer).

2) If `ifac=0` in one or more "turned on" zone specifications and one or more surface-oriented variables are specified in `ivarp`, the variables will be removed from the output dataset and a warning message will be echoed to the screen.

imin, imax, jmin, jmax, kmin, kmax – Specifies the extent of the desired extraction in the *ijk* directions. Numbering depends on the value of `interp` or the "shorthand" value as explained below.

`interp=1` (interpolate grid points to cell centers)

i, j, k min = cell # in that direction to start with
i, j, k max = cell # in that direction to end with +2

`interp=2` (interpolate flow data to grid points)

i, j, k min = grid point # in that direction to start with
i, j, k max = grid points # that direction to end with

shorthand values independent of `interp` setting

i, j, k min = cell or grid point # to start with
i, j, k max = -1 (extract all values in this direction)
-2 (extract all values in this direction less 1)
-3 (extract all values up to the midpoint in this direction)

***Tech Tip**: To extract data from a plane, set the min and max values in that direction to be the same.*

bkmin, bkmax – Specifies the range of master block numbers from which to extract data. Entering the shorthand value of "-1" in the `bkmax` flag tells POSTFLOW that the value of `bkmax` is the number of the last block in the simulation. For example, in a simulation composed of four master grid blocks:

```
bkmin=2          bkmax=-1
```

tells POSTFLOW to extract data from master blocks #2, #3, & #4.

zonetitle – An ASCII string surrounded by single or double quotes that will be used to name the zone if Tecplot output is specified. If a zone name is not desired, this flag should contain an empty string as shown below.

Tecplot output	"name of the zone"
Non-Tecplot output	""

I/O Filenames – All filenames must be enclosed with single or double quotes

fname – Specifies the name of the restart file to process. (*Required)*

pname – Specifies the name of the output file to create. (*Required)*

gname – Specifies the name of the grid file to process. (*Optional. Only needed if* `ingrid` *is > 0)*.

bname – Specifies the name of the boundary condition file to process. (*Optional. Only needed if* `inbcf` *> 0)*.

5.3 Neptune Sample Case

The sample case used throughout the DPLR Code User Manual to illustrate how the Code Package works describes a Neptune entry type probe with an ellipsoidal body as shown in Figure 5-2. This case is an example of aerocapture, where drag from the atmosphere is used to decelerate the vehicle and bring it into orbit.

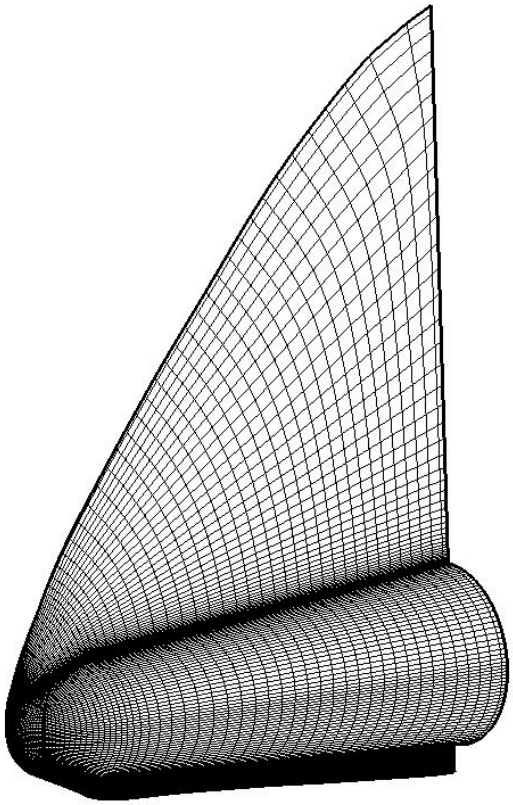


Figure 5-2 Neptune Probe

5.3.1 Neptune Input Deck

The input deck below shows two problem-specific entries to make for POSTFLOW to process a restart file generated during the DPLR simulation of the Neptune case. One input deck focuses on data generated at the surface of the probe (postsurf.inp) and one examines data on the pitchplane (postpitch).

```
Input file for postflow

imemmode itruev  ifstat
  2        1       0

inrest   ingrid  inbcf   ouform  iwrtd
 11        0       0       26      0

interp   nzones   isep    istyp   iunits
  1       10       0        1       1

lref   aref   xmc   ymc   zmc   imrx  imry  imrz
 1.0    1.0   0.0   0.0   0.0    0     0     0

iwind   cxs   cys   czs
  0     1.0   0.0   0.0

iexbc                  <== list of BC numbers to extract from dataset
 19

ivarp                  <== list of variable numbers to extract from
dataset
 0 110 120 154 150 151 152

Tecplot/plot3d zone information:
iwrt ifac imin imax jmin jmax kmin kmax  bkmin  bkmax  zonetitle
  0,   1,   1,   1,   1,  -1,   1,   1,    1,    -1    'stag2d'
  0,   2,   1,  -1,   1,   1,   1,   1,    1,    -1    'body2d'
  0,   0,   1,  -1,   1,  -1,   1,  -1,    1,    -1    'flow2d'
 -1,   0,   1,  -1,   1,  -1,   1,  -1,    1,    -1    'terminator'

fname,pname,(gname),(bname)
'neptune'
'postpitch'
```

Figure 5-3 POSTFLOW Input Deck for Pitchplane Analysis of Neptune Probe

```
nput file for postflow

imemmode itruev  ifstat
  2         1       0

inrest   ingrid  inbcf   ouform  iwrtd
 11         0       0       26      0

interp   nzones   isep   istyp    iunits
  1        10       0       1         1

lref   aref   xmc   ymc   zmc   imrx  imry  imrz
 1.0    1.0   0.0   0.0   0.0    0     0     0

iwind   cxs   cys   czs
  0     1.0   0.0   0.0

iexbc                   <== list of BC numbers to extract from dataset
 25,26

ivarp                   <== list of variable numbers to extract from
dataset
 0 110 120 507 521

Tecplot/plot3d zone information:
iwrt ifac imin imax jmin jmax kmin kmax  bkmin  bkmax  zonetitle
  0,   1,   1,   1,   1,  -1,   1,   1,    1,    -1    'stag2d'
  0,   2,   1,  -1,   1,   1,   1,   1,    1,    -1    'body2d'
  0,   0,   1,  -1,   1,  -1,   1,  -1,    1,    -1    'flow2d'
 -1,   0,   1,  -1,   1,  -1,   1,  -1,    1,    -1    'terminator'

fname,pname,(gname),(bname)
'neptune'
'postsurf'
```

Figure 5-4 POSTFLOW Input Deck for Surface Analysis of Neptune Probe

5.3.2 Neptune Input Deck Settings

The following table explains the meaning of the input deck settings in this sample case.

Input Flag	Setting	Explanation
imemmode	2	Run POSTFLOW in high memory mode so that all the features of the program are available.
itruev	1	Use accurate 2nd order expressions to compute derivative values.
ifstat	0	Process flow variables instantaneously.
inrest	11	The restart file to process is an XDR parallel archival file.
ingrid	0	Use grid information found in the restart file.
inbcf	0	Use boundary condition information found in the restart file.
ouform	6	Output post-processed data into a Tecplot point binary file.
iwrtd	0	Do not reconstruct DPLR input decks from the restart file for storage in a subdirectory of your working directory.
interp	1	Interpolate grid points to cell centers.
nzones	10	The maximum number of output data zones to be generated is 10.
isep	0	All active output datasets are written to a single file.
istyp	1	This flag is ignored in DPLR 4.01.0.
iunits	1	Include SI units in the output file
lref	1.0	Use 1 meter as the reference length to normalize moment coefficients.
aref	1.0	Use 1 square meter as the reference area to normalize force and moment coefficients
xmc	0.0	The moment reference center is located at 0.0 on the x axis.
ymc	0.0	The moment reference center is located at 0.0 on the y axis.
zmc	0.0	The moment reference center is located at 0.0 on the z axis.
imrx	0	Do not enforce symmetry about the *yz* plane.

Input Flag (cont)	Setting (cont)	Explanation (cont)
imry	0	Do not enforce symmetry about the *xz* plane.
imrz	0	Do not enforce symmetry about the *xy* plane.
iwind	0	Do not alter the raw output data (Ignored).
cxs	1	Cosine of the global wind axis in the x direction = 1 (Ignored)
cys	0	(Ignored)
czs	0	(Ignored)
iexbc	19 25 26	Extract the boundary conditions from the z symmetry plane (pitch plane). Extract data from the intersection between a catalytic isothermal wall (ibc=25) and a catalytic radiative equilibrium wall (ibc=26) i.e. the probe surface.
ivarp	0	Extract all grid coordinates.
	110	Extract pressure data.
	120	Extract translational temperature data.
	150	Extract velocity in the x-direction (*u*)
	151	Extract velocity in the y-direction (*v*)
	152	Extract velocity in the z-direction (*w*)
	154	Extract the frozen Mach number.
	507	Extract the total wall shear stress (tau)
	521	Extract total wall heat transfer.
iwert	0	Do not extract data in these zones
fname	neptune	The restart file to be post-processed by POSTFLOW is named 'neptune'
pname	postsurf postpitch	The output files created by POSTFLOW for use by Tecplot are named 'postsurf' and 'postpitch'.

5.3.3 Neptune Output Summary

Upon execution, POSTFLOW will create an on-screen summary of the problem for each input deck run as shown below:

```
 postflow
 NASA Ames Version 4.01.0
 Maintained by Mike Wright;  last modified: 02/05/09
 ******************************************


Parsing the restart file to get physical modeling data...
 restart file format: NASA Ames Version 4.01.0
 solution run at: Thurs Feb  5 08:18:10 2009

 run in   500 iterations in 4.23E+03 seconds

 CPP-macro settings enabled during run:
       AMBIPOLAR=1
       PARKTEXP=0.50
       NOHTC

 Keq limiter set at  100.00

 input  ns =  5;  ner =  0;  nev =  0;  net =  0
  number of blocks =   2
 file dimension   =  3

 extracting the following BCs :    17   18   19
 note that extraction of pointwise BCs not supported yet

 output variables=x,y,z,p,T,M,u,v,w

 running in high memory mode

 interpolating grid to cell centers

 processing grid variable  1  2  3
 processing flow variable  1  2  3  4  5  6  7  8  9 10

 block #  1: nx =   32; ny =   16; nz =   64
      zone t=BC19         i=  34 j=   1 k=  66

 processing grid variable  1  2  3
 processing flow variable  1  2  3  4  5  6  7  8  9 10

 block #  2: nx =   48; ny =   64; nz =   64
      zone t=BC19         i=  50 j=   1 k=  66

      zone t=BC19         i=  50 j=   1 k=  66


 writing tecplot file: postpitch.dat

 using grid file: neptune-8PE.pgrx
 using flow file: neptune.pslx
```

Figure 5-5 POSTFLOW Onscreen Summary for Pitchplane Analysis of Neptune Probe

```
 ******************************************
 postflow
 NASA Ames Version 4.01.0
 Maintained by Mike Wright;  last modified: 02/05/09
 ******************************************


Parsing the restart file to get physical modeling data...
 restart file format: NASA Ames Version 4.01.0
 solution run at: Thurs Feb  5 08:18:10 2009

 run in   500 iterations in 4.23E+03 seconds

 CPP-macro settings enabled during run:
       AMBIPOLAR=1
       PARKTEXP=0.50
       NOHTC

 Keq limiter set at  100.00

 input  ns =  5;  ner =  0;  nev =  0;  net =  0
 number of blocks =   2
 file dimension   =  3

 extracting the following BCs :    25   26
 note that extraction of pointwise BCs not supported yet

 output variables=x,y,z,p,T,tau,qw

 running in high memory mode

 processing grid variable  1  2  3
 interpolating grid to cell centers

 processing flow variable  1  2  3  4  5  6  7  8  9 10

 block #  1: nx =   32; ny =   16; nz =   64
  ==> extracted derivative data from the KMIN-surface
  ==> derivative data computed using full viscous fluxes
      zone t=BC26         i=  34 j=  18 k=   1

 processing grid variable  1  2  3
 processing flow variable  1  2  3  4  5  6  7  8  9 10

 block #  2: nx =   48; ny =   64; nz =   64
  ==> extracted derivative data from the KMIN-surface
  ==> derivative data computed using full viscous fluxes
      zone t=BC26         i=  50 j=  66 k=   1


 writing tecplot file: postsurf.dat

 using grid file: neptune-8PE.pgrx
 using flow file: neptune.pslx
```

Figure 5-6 POSTFLOW Onscreen Summary for Surface Analysis of Neptune Probe

5.3.4 Neptune Output Information

In addition to verifying the values entered into the POSTFLOW input deck, the POSTFLOW output summary displays information about the grid and the flow solution components of the restart file being extracted for processing.

In this sample case, POSTFLOW displays the Keq (equilibrium constant) limiter that DPLR calculated from `ikeq` setting in the DPLR input deck, restates that the number of species used in the simulation was 5, confirms that the rotational, vibrational, and translational energies of the flow were ignored for this simulation, and verifies that this solution for the Neptune entry probe is based upon a 2 block, 3D simulation. POSTFLOW then names the output variables that were specified by `ivarp` in the input deck and confirms that post-processing of the restart file is taking place in high memory mode.

Next, POSTFLOW displays a running indicator of block-by-block progress in processing the restart file, while verifying block dimensions.

Finally, the summaries show the name of the output files, in this case `postpitch.dat and postsurf.dat`, while displaying the names of the grid file and the restart file that were used to create the output.

5.4 Extracting Datasets

The primary use of POSTFLOW is to extract volume or surface data from the restart file for further post-processing or visualization.

Using the `ouform` flag in the POSTFLOW input deck, data can be saved in two primary output file formats:

- `plot3d (ouform = 2, 3, 22, 23, 32, 33)`
- Tecplot `(ouform = 5, 6, 25, 26)`

The `plot3d` format is a standard CFD output format that can be read by most commercial post-processing tools while the Tecplot format is specific for use with Amtec's Tecplot data visualization software.

POSTFLOW can write Tecplot ASCII (`"*.dat"`) files as well as binary (`"*.plt"`) files, although Tecplot binary output requires linking to the Amtec-provided "tecio.a" (or "tecio64.a") runtime library. If this library is not available on your machine, Tecplot binary files cannot be generated.

`Gzipped plot3d` output (`ouform=32,33`) is generated via a system call to the ***gzip*** utility provided with UNIX and LINUX systems. This option may not be available on Windows systems.

5.4.1 Volume Data

Volume data can be extracted from a restart file using the zone specification lines in the POSTFLOW input deck.

For example, assume that a simulation was performed on a five-block, 3D volume grid, and the desired output variables are pressure (`ivarp = 110`), temperature (`ivarp = 120`), Mach number (`ivarp = 154`), and pointwise residual (`ivarp = 999`). The `ivarp` array would be:

```
ivarp
110 120 154 999
```

The following zone specification lines could then be used to extract data from the entire volume:

```
iwrt ifac imin imax jmin jmax kmin kmax  bkmin  bkmax  zonetitle
 1,   0,    1,  -1,   1,  -1,   1,  -1,    1,    -1    'volume'
-1,   0,    1,  -1,   1,  -1,   1,  -1,    1,    -1 'terminator'
```

Using the shorthand code of -1 to mean "maximum" or "all", these lines tell POSTFLOW to read the "volume" line (`iwrt=1`), ignore surface data (`ifac=0`), extract all points in the *i* (`imin=1, imax=-1`), *j* (`jmin=1, jmax=-1`) and *k* (`kmin=1, kmax=-1`) directions from all master blocks (`bkmin=1, bkmax=-1`). Then, with `iwrt=-1`, the terminator line tells POSTFLOW to stop reading zone specification information.

POSTFLOW will now generate five output zones (one for each block) which contain the entire volume. Each zone will be called "volume" if a Tecplot output file format is selected by setting `ouform=5:6,25:26` in the POSTFLOW input deck.

5.4.2 Surface Data

Surface data can be extracted from a restart file in two ways:

- Using zone specification lines
- Using the `iexbc` flag

Zone Specification Lines

Continuing with the 5 block 3D example in Section 5.4.1, assume that all blocks have a body surface at *j* = 1, and that these five surfaces completely define the body. The following zone specification lines could then be used to extract data from the entire body surface:

```
iwrt ifac imin imax jmin jmax kmin kmax  bkmin  bkmax  zonetitle
 1,   2,    1,  -1,   1,   1,   1,  -1,    1,    -1    'body'
-1,   0,    1,  -1,   1,  -1,   1,  -1,    1,    -1 'terminator'
```

Using the shorthand code of -1 to mean "everything" or "all", these lines tell POSTFLOW to read the "body" line (`iwrt=1`), extract the *j* face (`ifac=2`), and extract the *j* surface (jmin=1, jmax=1) from all blocks (`bkmin=1, bkmax=-1`). Then, with `iwrt=-1`, the terminator line tells POSTFLOW to stop reading zone specification information.

Now, assume further that the exit (outflow) plane of the problem can be completely defined as the `imax` surface of block #5. You can tell POSTFLOW to extract data from this surface by creating the following zone specification lines:

```
iwrt ifac imin imax jmin jmax kmin kmax  bkmin  bkmax  zonetitle
 1,   1,   -1,  -1,   1,  -1,   1,  -1,    5,     5    'outflow'
-1,   0,    1,  -1,   1,  -1,   1,  -1,    1,    -1 'terminator'
```

These lines tell POSTFLOW to read the "outflow" line (`iwrt=1`), extract the *i* face (`ifac=1`), and extract the *i* surface (imin=-1, imax=-1) from block #5 (`bkmin=5, bkmax=5`). Then, with `iwrt=-1`, the terminator line tells POSTFLOW to stop reading zone specification information.

iexbc Flag

Instead of using zone specification lines to extract surface data from a restart file (a process that can be cumbersome to set up and which requires you to pre-determine the locations of all surface sub-zones in the simulation), you can use the `iexbc` flag to accomplish the same result.

To extract data from all six surfaces of each master block in the simulation, simply set `iexbc` to one or more values of the boundary condition settings allowed for the `ibc` flag in the DPLR Input deck (See Section 4.2).

For example, if you want to extract `ivarp`-specified data from all possible symmetry planes and outflow boundaries of a multiblock grid, the `iexbc` setting would be as follows:

```
iexbc
17 18 19 3
```

This setting tells POSTFLOW to extract the `ivarp`-specified variables for the *x, y, z* planes of symmetry and the first order extrapolation of the supersonic exit surface.

Extracting data via the `iexbc` flag is a powerful tool within POSTFLOW and should be used whenever possible to simplify extraction of complex surface datasets.

Tech Tip: *Note that the* `iexbc` *flag can be used together with the zone specification lines in a single POSTFLOW run to extract BOTH surface and volume datasets. By using a combination of these methods, it should be possible to extract almost any desired subset of flowfield data.*

5.4.3 Line Data at the Intersection of Two Boundaries

The `iexbc` flag can also be used to extract data at the intersection of two surfaces, such as along the vehicle centerline. To extract surface intersections, specify your two desired boundary conditions and separate them with a forward slash.

For example, if you want to extract quantities on a radiative equilibrium catalytic surface (`ibc=26`) along the *xz*-symmetry plane `(ibc=18)` you would enter:

```
iexbc
26/18
```

The first number is always the "reference" boundary, telling POSTFLOW how to extract desired derivative quantities (such as heat flux and shear stress). The second number is the boundary condition that you want to intersect with the reference boundary.

You can request multiple intersections in a single POSTFLOW run if you present them in a space or comma-separated list. For example, the following `iexbc` entry will extract intersections between a radiative equilibrium catalytic surface (26) and all 180° symmetry planes:

```
iexbc
26/17 26/18 26/19
```

You can extract boundary intersection data in conjunction with extracting standard boundary conditions and volume data. For example:

```
iexbc
26/18 3
```

would extract the intersection between boundary condition 26 and 18 as well as data for the exit plane.

Tech Tip: *Intersection extraction does not currently work properly with pointwise specified boundary conditions. However, you can extract the intersection with all pointwise specified boundary conditions by using* `ibc = 0` *in an intersection specifier.*

5.4.4 Zone Minima or Maxima

POSTFLOW can extract the minimum or maximum values of selected output variables in each output dataset, and, if desired, the *ijk* location of these values.

You can accomplish this by setting the value of `ouform` in the POSTFLOW input deck either to 7 or to 17. In both cases, however, the results of this operation are only written to the screen in the standard out, (STOUT) not to an output datafile.

For example, if you set `ouform=7` in the Neptune Sample Case described in Section 5.3, the on-screen output summary would be show:

```
block # 1: nx =   32; ny =   16; nz =   64
      zone t=BC19          i=  34 j=   1 k=  66

 Zone Maximum and Minimum Values:
 p       [max] =  5.0043E+04;   [min] =  3.5910E+01
 T       [max] =  1.5345E+04;   [min] =  1.2807E+02
 M       [max] =  3.2322E+01;   [min] =  0.0000E+00

 processing grid variable  1  2  3
 processing flow variable  1  2  3  4  5  6  7  8  9

 block # 2: nx =   48; ny =   64; nz =   64
      zone t=BC19          i=  50 j=   1 k=  66

 Zone Maximum and Minimum Values:
p       [max] =  4.4431E+04;   [min] =  3.5910E+01
T       [max] =  1.4203E+04;   [min] =  1.2807E+02
M       [max] =  3.2322E+01;   [min] =  0.0000E+00
```

If you set `ouform = 17`, POSTFLOW displays a longer listing to this onscreen summary which includes the *ijk* locations of these maximum and minimum values in the zone.

Tech Tip: *Note that the ijk location is computed relative to the output zone. If ijk values for all blocks are required, the entire volume should be selected as output.*

5.4.5 Integrated Surface Data

POSTFLOW can integrate data for the following surface variables:

- face area (ivarp = 23)
- total heating (ivarp = 531)
- mass flow rate (ivarp = 594)
- thrust (ivarp = 596)
- aerodynamic forces (ivarp = 600:673
- aerodynamic moments (ivarp = 700:773)
- species mass flow rate (ivarp = 5000+*n*)

You can accomplish this by setting `outform=8` and `interp=11` and making sure that all output datasets define surfaces, either with the `iexbc` or the `ifac flag.` As with the computation of minimum and maximum values, the results of this operation are only written to the screen in the standard out (STOUT) where results for each zone and a sum for all zones are shown, not to an output datafile.

For example, if you set `ouform=8 and interp=11` in the Neptune Sample Case described in Section 5.3, the on-screen output summary might show:

```
block # 1: nx =   32; ny =   16; nz =   64
  ==> extracted derivative data from the KMIN-surface
  ==> derivative data computed using full viscous fluxes
      zone t=BC19         i=  32 j=  16 k=   1

        Fx      =  9.872234694840E+02   (N)
        Fy      =  3.249055280159E+02   (N)
        Fz      = -3.865734146780E+02   (N)

 processing grid variable  1  2  3
 processing flow variable  1  2  3  4  5  6  7  8  9

 block # 2: nx =   48; ny =   64; nz =   64
  ==> extracted derivative data from the KMIN-surface
  ==> derivative data computed using full viscous fluxes
      zone t=BC19         i=  48 j=  64 k=   1

        Fx      =  2.919904481514E+03   (N)
        Fy      =  1.605495325835E+04   (N)
        Fz      = -9.258734449289E+03   (N)
```

```
Integrated Surface Quantities
Summary Over All Output Surfaces:
XZ-Symmetry Enforced During Final Summation

       Fx      =  7.814255901995E+03   (N)
       Fy      =  0.000000000000E+00   (N)
       Fz      = -1.929061572793E+04   (N)
```

Tech Tips:

1) Any `ivarp` *values not included in the list above will be removed from the input deck when ouform=8.*

2) If aerodynamic forces are selected and `iwind` *is set to either 1 or 2, output forces will be rotated into the wind coordinate system based on either the internal* `(iwind = 1)` *or provided* `(iwind = 2)` *velocity cosines, and will be output as lift, drag, and side forces in addition to the xyz forces otherwise reported. Note that this option assumes that the employed grid is in standard aircraft coordinates.*

5.4.6 Freestream Data

POSTFLOW can extract freestream data from the restart file.

You can accomplish this by setting the value of `ouform` in the POSTFLOW input deck either to 10 to display requested ivarp values with their SI units or to 110 to display a tabular listing of data better suited for direct import to a spreadsheet application. In both cases, however, the results of this operation are only written to the screen in the standard out, (STOUT) not to an output datafile.

Freestream data are calculated and output for each grid block in the simulation, irrespective of any surface extraction or zone specification flags that have been set. Separate freestream data are presented for each grid block, since DPLR allows multiple freestream specifications to be applied when a simulation is run. However, in most cases, all blocks will have the same freestream information.

For example, if you set `ouform=10 and ivarp=110,120,154,58` in the Neptune Sample Case described in Section 5.3, the on-screen output summary might show:

```
block # 1: nx =   32; ny =   16; nz =   64

 Freestream Quantities:

     Block # 1

        p       =  3.591044259306E+01   (Pa)
        T       =  1.280700000000E+02   (K)
        M       =  3.232180261501E+01   ()
```

```
        Re/L    =  3.151858720834E+05   (1/m)

 block # 2: nx =   48; ny =   64; nz =   64

     Block # 2

        p       =  3.591044259306E+01   (Pa)
        T       =  1.280700000000E+02   (K)
        M       =  3.232180261501E+01   ()
        Re/L    =  3.151858720834E+05   (1/m)
```

5.4.7 Extracting Data for External Codes

Several of the output format options (`ouform`) in POSTFLOW create files intended for use with third-party codes or provided post-processing utilities. In these cases, options are hardwired to the values required by the particular third party software.

Extraction for Moment

As of release version 3.05, POSTFLOW can directly compute moments or moment coefficients. However, the ***Moment*** utility, provided as part of the DPLR Code Package, can also do this computation. ***Moment*** requires plot3d grid and function files along with an input "moment.inp" file to be created by POSTFLOW (See Section 9.1.6).

To tell POSTFLOW to create these files:

- set ouform=11
- set interp=11
- set ivarp to either total forces (604:606), pressure forces (614:616), or viscous forces (624:626)
- set output datasets to define surfaces either with iexbc or ifac

At the current time the only function of ***Moment*** that is not built into POSTFLOW is for the extraction of hinge moments.

Extraction for *RADEQUIL*

To tell POSTFLOW to output a line-of-sight file for further processing with the shock layer radiation code ***RADEQUIL***, set ouform=28.

POSTFLOW will then automatically assume the following (hardwired) settings:

interp = 1

ivarp = 11 12 13 110 120 125 1600

The output will be an ASCII format file with the suffix ".los".

When using this option, you must specify "single body-normal line of sight" either with the `iexbc` flag (e.g., `iexbc = 14` will extract the stagnation line of an axisymmetric body), or by specifying a 1D line for extraction with the Tecplot specifier flags.

Because ***RADEQUIL*** requires a certain set of species mole fractions in a certain order, POSTFLOW will compare the input mole fractions with the expected set in ***RADEQUIL*** and reorder as necessary. Species expected by ***RADEQUIL*** that are not in the current CFD dataset will be filled in with zeros as required. The resulting file should then be ready for direct processing in ***RADEQUIL***.

5.4.8 NaN's (Not A Number)

POSTFLOW can extract the locations of any NaN's in the restart file to help you determine where the simulation begins to diverge. Although rarely used in practice, this option can be a handy tool to use in locating the occasional evil bug.

You can accomplish this by setting `ouform=18`.

The output data generated by this operation consists of a list of *ijk* locations of all NaN's in the volume, listed block-by-block. However, the results are only written to the screen in the standard out, (STOUT) not to an output datafile.

Tech Tip: *Note that once a NaN is generated by DPLR, it will quickly be convected throughout the solution domain, so if you want to view the location where the NaN first occurred, you need to stop the simulation and write a restart file at the conclusion of the iteration in which the NaN was first generated, typically the iteration PRIOR to when the residual itself becomes NaN.*

Chapter 6 - DPLR Input / Output Files

Contents

6.0 Introduction

This chapter of the DPLR Code User Manual discusses the 11 types of input and output files created and/or used by the DPLR Code Package Version 4.01.1: grid files, zonal interface files, boundary condition files, runtime control files, restart files, chemistry files, radiation files, convergence files, aerodynamic files, log files and Tecplot files.

Although each file type can be written in two or more different formats, not all formats are compatible with all parts of the DPLR Code. For example, FCONVERT can read plot3d formatted grid and function files as input, but DPLR2D, DPLR3D, and POSTFLOW cannot. (See Section 9.2 for more information on file formats.)

6.1 Grid Files

Grid files define the discretized computational geometry of the CFD problem. Grid files can exist in the following formats:

Description	Suffix
unformatted parallel	pgrd
XDR parallel	pgrx
ASCII parallel	pgra
unformatted plot3d	gu
XDR plot3d	gx
ASCII plot3d	g
gzipped ASCII plot3d	gz

The plot3d files created by third-party grid-generation software packages such as ***GridGen*** or ***GridPro*** can be read as input by FCONVERT, which typically converts them to the XDR parallel grid file format "*.pgrx" to be used in a DPLR simulation run.

Tech Tip: *Although FCONVERT can write grid files in all of the formats listed above, the preferred format for use in the DPLR working environment is XDR parallel* `("*.pgrx")` *- a binary, machine-readable file.*

6.2 Zonal Interface Files

A zonal interface, or zonal boundary, is a region where two grid blocks abut, sharing the same grid points. Information about these areas of abutment, in the form of an ASCII zonal interface file, must be provided as input for DPLR to ensure that data are mapped correctly across grid blocks during the CFD computation.

Zonal interface files can be prepared in three ways:

- manually, through direct observation of the serial plot3D input grid (init=1)
- automatically, by FCONVERT (init=2-4
- automatically, by using the TEMPLATE utility (See Section 8.1.5)

When the plot3D input grid describes a relatively simple set of master blocks and resulting zonal boundaries, developing the information for a zonal interface file by hand may be a straightforward way to proceed. However, when a multi-block input grid contains a large number of blocks or describes complex geometries (as is often the case in aerospace problems), using FCONVERT or TEMPLATE to develop the detailed data required to accurately describe each zonal interface is likely to be the more productive approach.

6.2.1 Creating Zonal Interface Files by Hand

Step 1: **Examine** the plot3D input grid to determine the location of all zonal interfaces, making note of how the points in each master block abut those in another block.

Step 2: **Open** the text editor program for your system, and create an ASCII file with the format shown below.

```
ZONAL BOUNDARY INFORMATION
Cell Matching - No dummy cells
----------------------------------------
zvers izdum

nblk ninta nintc

----------------------------------------
Zonal Boundary #
nz nface ndr1 nst1 nen1 ndr2 nst2 nen2
```

Step 3: **Enter** values that describe the number, location, extent direction, and range of each zonal interface identified in the input plot3D grid file.

Step 4: **Save** the file.

Action: At the command line, type:

```
save 'filename.inter'
```

Result: The ASCII zonal interface file required by DPLR (and identified by the `xname` flag in the FCONVERT input deck) is saved.

6.2.2 Input Variables for Zonal Interface Files

Input variables required in a zonal interface file are discussed below in the order they appear in the file.

`zvers` – Specifies the version number of the interface file. This is used by FCONVERT to automatically upconvert older interface files when they are read, thus assuring full backward compatibility. Allowable values are the real numbers of the major and minor releases of the DPLR Code Package, from 2.31 through the current version number, 4.01.1.

`izdum` – Specifies whether dummy cells are accounted for in the interface file. Allowable values are:

0	Input file does not include dummy cells
1	Input file includes dummy cells

> ***Tech Tip****: This option is meant for developers to use in debugging. If* `izdum=1`*, FCONVERT will automatically strip the dummy cell information before processing the zonal interface file, which could have unwanted results.*

`nblk` – Specifies the number of master blocks in the input plot3D grid.

`ninta` – Specifies the number of zonal interfaces in the input plot3D input grid.

nintc – Specifies the number of corner/edge zonal interfaces in the input plot3D grid. Allowable values are:

0 Input grid contains no corner/edge zonal interfaces.

Non-zero value – meant for debugging

***Tech Tip**: This option is meant for developers to use in debugging. If `nintc>0`, FCONVERT will automatically strip the corner/edge zonal interface information before processing the file.*

nz – Specifies the grid blocks that define the common face of the zonal boundary being described.

nface – Specifies the block faces that abut, thereby indicating the plane in which the zonal boundary lies. Allowable values are:

1 *imin* face

2 *imax* face

3 *jmin* face

4 *jmax* face

5 *kmin* face

6 *kmax* face

***Tech Tip**: If the grid is for a 2D or axisymmetric problem, `nface` must =1-4, since such problems are assumed to lie in the ij plane.*

ndr1 – Specifies the first extent direction of the zonal boundary being described. Allowable values are:

1 *i*-direction

2 *j*-direction

3 *k*-direction

nst1 – Specifies the starting point of the interface range (from cell center to cell center) in the direction indicated by the value in ndr1.

nen1 – Specifies the ending point of the interface range (from cell center to cell center) in the direction indicated by the value in ndr1.

ndr2 – Specifies the second extent direction of the zonal boundary being described. Allowable values are:

1 *i*-direction

2 *j*-direction

3 *k*-direction

nst2 – Specifies the starting point of the interface range in the direction indicated by the value in ndr2.

nen2 – Specifies the ending point of the interface range in the direction indicated by the value in ndr2.

6.2.3 Neptune Zonal Interface File

The following zonal interface file was created by hand for the Neptune sample case (inint=1).

```
ZONAL BOUNDARY INFORMATION
Cell Matching - No dummy cells
 ----------------------------------------
   zvers izdum
   3.05    0

   nblk ninta nintc
    2      3     0
----------------------------------------
Zonal Boundary #  1
  nz  nface ndr1  nst1  nen1 ndr2 nst2  nen2
   1     1     2     1    16    3     1     64
   2     1     2    16     1    3     1     64

----------------------------------------
Zonal Boundary #  2
  nz  nface ndr1  nst1  nen1 ndr2 nst2 nen2
   1     2     2     1    16    3     1     64
   2     1     2    49    64    3     1     64
----------------------------------------
```

```
Zonal Boundary #  3
  nz  nface ndr1 nst1   nen1 ndr2 nst2 nen2
   1     3     3     1     64    1      1    32
   2     1     3     1     64    2     17    48
=======================================
```

6.2.4 Input Values in Neptune Zonal Interface File

The input plot-3D grid for this problem consists of two master blocks and three interfaces between these two blocks. Each interface (zonal boundary) is described by two lines of data in the interface file. The following table explains the meaning of the values entered into the zonal interface file for this sample case.

Input Value	Setting	Explanation
zvers	3.05	The 3.05 version of the DPLR Code package is being used.
izdum	0	There are no dummy cells accounted for in this interface file.
nblk	2	There are 2 master blocks in the input grid file.
ninta	3	There are 3 zonal interfaces in the input grid file.
nintc	0	Input grid contains no corner/edge zonal interfaces.
nz	1,2; 1,2; 1,2	Master blocks #1 and #2 participate in the three zonal boundaries being described.
nface	1,1; 2,1; 3,1	The first zonal interface is located at the imin of both blocks, placing it in the kj plane. The second zonal interface is located at the imax of one block and the imin of the other, placing it in the jk plane. The third zonal interface is located at the jmin of one block and the imin of the other, placing it in the ik plane,
ndr1	2,2; 2,2; 3,3	The first extent direction for the first interface zone is j. The first extent direction for the second interface zone is also j. The first extent direction for the third interface zone is k.
nst1	1, 16; 1,49; 1,1	The starting point in the first extent direction for one block in the first zonal interface is 1 and for the abutting block it is 16. The starting point in the first extent direction for one block in the second zonal interface is 1 and for the abutting block it is 49. The starting point in the first extent direction for one block in the third zonal interface is 1 and for the abutting block it is also 1.

Input Value (cont)	Setting (cont)	Explanation (cont)
nen1	16,1; 16,64; 64,64	The ending point in the first extent direction for one block in the first zonal interface is 16 and for the abutting block it is 1. The ending point in the first extent direction for one block in the second zonal interface is 16 and for the abutting block it is 64. The ending point in the first extent direction for one block in the third zonal interface is 64 and for the abutting block it is also 64.
ndr2	3,3; 3,3; 1,2	The second extent direction for the first interface zone is k. The second extent direction for the second interface zone is also k. The second extent direction for the third interface zone is i to j.
nst2	1, 1; 1,1; 1,17	The starting point in the second extent direction for one block in the first zonal interface is 1 and for the abutting block it is also 1. The starting point in the second extent direction for one block in the second zonal interface is 1 and for the abutting block it is also 1. The starting point in the second extent direction for one block in the third zonal interface is 1 and for the abutting block it is 17.
nen2	64,64; 64,64; 32,48	The ending point in the second extent direction for one block in the first zonal interface is 64 and for the abutting block it is also 64. The ending point in the second extent direction for one block in the second zonal interface is 64 and for the abutting block it is also 64. The ending point in the second extent direction for one block in the third zonal interface is 32 and for the abutting block it is 48.

6.2.5 Creating Zonal Interface Files Automatically

Although it is valuable to fully understand the meaning and origin of data in the zonal interface files, it is likely that day-to-day use of the DPLR Code Package will more often involve automatic generation of these files.

Currently, there are three tools available to automatically compute and generate zonal interface files that are readable by DPLR:

1. GASP® / zbconvert
2. Template
3. FCONVERT

GASP / zbconvert

Some commercial grid generation tools are capable of automatically generating interface information in a format that is readable by the commercial CFD code ***GASP Version 3.*** For this reason, a utility (***zbconvert***) is included with the DPLR Code

Package that can convert zonal interface information from *GASP® Version 3* to DPLR –readable zonal interface files. See Section 9.1.1 for more information about the utility ***zbconvert***.

Template

Zonal interface files can also be created by the software tool ***Template***, developed by Scott Thomas and David Saunders, which automatically generates a DPLR input deck and interface file from a multi-block grid. ***Template*** is supplied as a utility with the DPLR 4.01.1 Code Package. (For more information about Template, see Section 9.1.5.)

FCONVERT

As previously discussed in Section 3.2, setting `inint`=2-4 in the FCONVERT input deck tells the program to automatically generate the type of zonal interface data required by DPLR for grid processing. However, each `inint` setting option offers different levels of computational speed and accuracy.

Setting `inint=2` results in a rapid detection of full-face zonal interfaces as shown in Figure 6-1 by comparing the centroid of each master block face (where the centroid is computed by averaging all cells in that face). Index directions of the two faces can be arbitrary as long as the centroids of a face pair are within a tolerance (determined internally based on grid dimensions and clustering). Because this method detects full-face interfaces only, this option should only be used if it is known that the input grid does not contain sub-face interfaces, i.e., areas where one block face abuts only a portion of another block face as shown in Figure 6-2. It is useful to note that the computational accuracy of this setting is comparable to that achieved using the ***Template*** software utility.

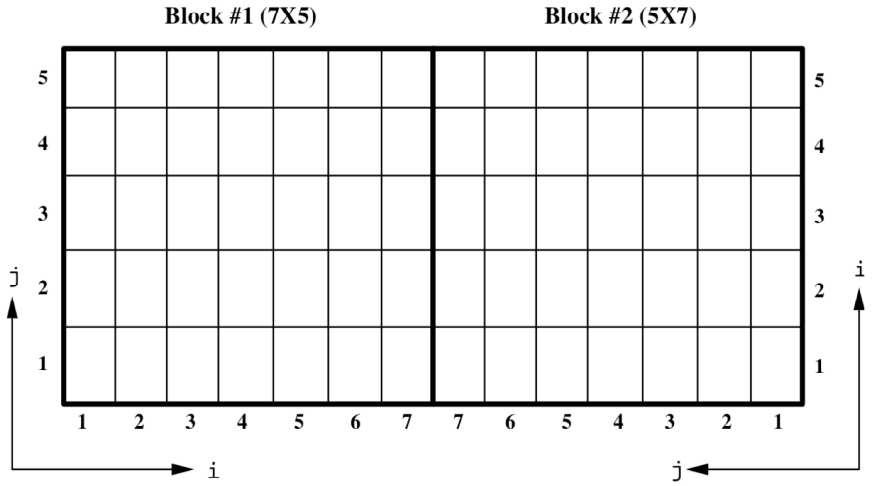


Figure 6-1 Full Face Zonal Interface

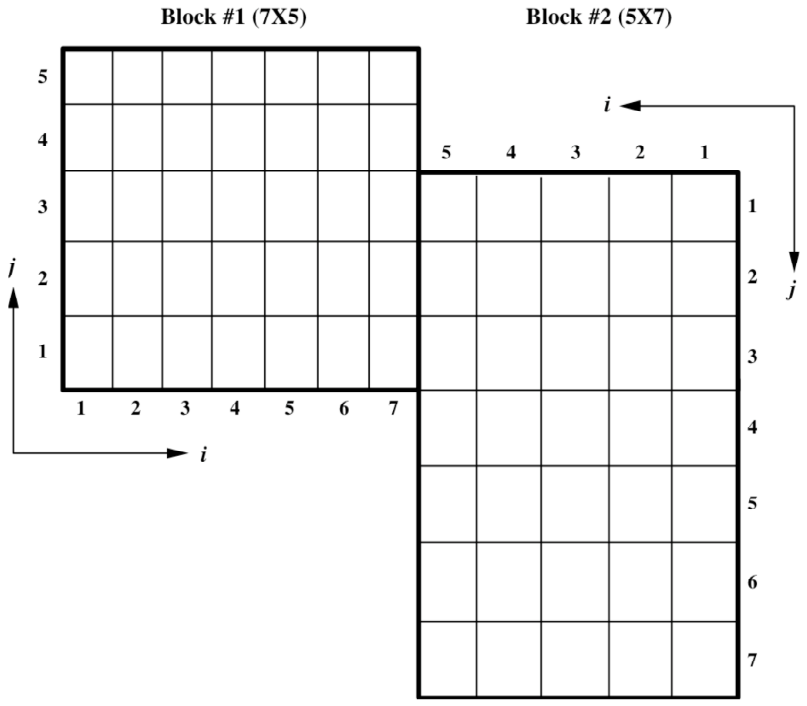


Figure 6-2 Sub-Face Zonal Interface

Setting `inint=3` results in moderately rapid detection of both full-face and sub-face zonal interfaces and is the recommended option for most DPLR cases. Using a modern (2007-era) computer and working at a speed of approximately one minute per million grid cells analyzed, this option works by examining all edge cells of all block faces for interfaces. Figure 6-3 shows a typical block where edge cells that are checked are highlighted in red. The only type of interface that will not be detectable using the `inint=3` option is a case where an interior sub-face of a 3D block face touches another interior sub-face of the same block as shown in Figure 6-4.

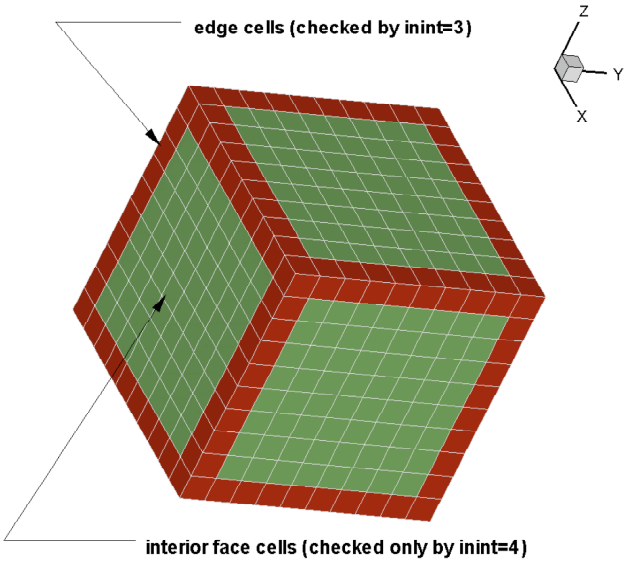


Figure 6-3 Edge Cells Checked with inint = 3 (shown in red).

Setting `inint=4` results in accurate detection of all zonal interfaces, including the interior-to-interior zonal boundaries that a setting of `inint=3` would miss as shown in Figure 6-3. As might be expected, this setting employs a slow search algorithm where every single exposed face cell is compared with every other exposed face cell, requiring on the order of 15 minutes of 2007-era computer time for every million grid cells analyzed. Cases requiring the use of this option are likely to be infrequent, although it is a useful tool to have when needed.

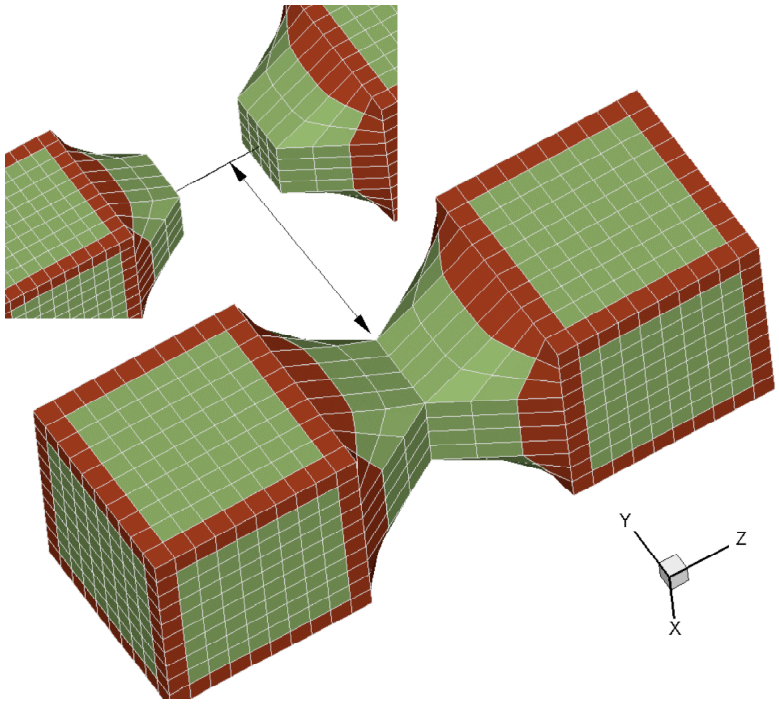


Figure 6-4 Interior-to-Interior Zonal Interface Detectable Only with inint = 4.

Tech Tips:

1) All intra-zonal interface boundaries (singularity or degenerate axes, self-closing blocks, etc.) will be detected using any of the detection options `inint` *= 2-4.*

2) Some grid generation programs and post-processors introduce round-off error in the xyz grid coordinates that can result in the points on either side of an interface being slightly different. FCONVERT has a built-in tolerance factor to determine when two slightly different points are likely the same, but this is not foolproof. To determine if all interfaces have been accurately detected, you can (1) compute them all by hand as a check case, or (2) run the resulting case and look for mismatched interfaces in the resulting solution, or (3) monitor the tolerance of each interface found in the input grid as reported by FCONVERT or (4) look for a large number of small patchy interfaces between two faces resulting from an FCONVERT run .

6.3 Boundary Condition Files

Boundary condition files (also called "pointwise boundary condition files") provide information to DPLR about the chemical, radiation, and turbulence conditions that exist at each point on a specified face of a master grid block.

Pointwise boundary condition files are optional. If you prepare one for your simulation, you must enter the filename in the `bname` flag in the DPLR input deck. Boundary condition files typically have the suffix `"*.pbca".` If a boundary condition file is not prepared for your simulation, you should set `bname` =`none`.

A generic boundary condition file named `pointwise.pbca` is distributed with the DPLR Code Package Version 4.01.1 can be found in the `cfdinput` directory.

6.3.1 Creating a Pointwise Boundary Condition File

Step 1: **Open** the text editor program for your system.

Action: At the command line prompt, type:

/[path to your cfdinput directory]/ `pointwise.pbca`

Result: A generic input file appears on screen, with place-holder default values as shown below. To customize the file for your simulation, remove the default values but take special care to preserve the line spacing. Specifically, there must be three lines (shown with # signs) between lines with value entries.

Step 2: **Enter** appropriate, problem-specific values for the input variables as described in Section 6.3.2.

Step 3: **Save** your boundary condition file to your working directory.

Tech Tip: *Although you can add as many lines as you need to specify the sizes and* `ibc` *numbers for each master block in your input grid, preserve the line spacing within each section and throughout the global areas of the input deck. If lines are added to or subtracted inappropriately within these areas, DPLR will not be able to read the file accurately.*

```
-2  4.01  0         #Pointwise PBCA File template, Version 4.01
#
# nblk  idim
#
     2   3
#
#  neq  ns ner nev nee net
#
    13   8   0   1   0   0
#
#  nmc nme nmt nmv  f2  f3  f4
#
     0   0   0   0   0   0   0
#
#  block sizes
#
    16  12  78
    40  40  78
#
#  ibc numbers for each block
#
    20  20  18  20  26  60
    20   3  18  18  26  60
#
#  Profile Data for Block #  1; Face #  6
#
  1.632708000000000E-01   1.632708000000000E-01   1.632708000000000E-01
  and so on...
#
#  Profile Data for Block #  2; Face #  6
#
  1.632708000000000E-01   1.632708000000000E-01   1.632708000000000E-01
  and so on...
# End PBCA Data
```

6.3.2 Input Flags for Pointwise Boundary Condition Files

Input flags for a pointwise boundary condition file are discussed below in the order they appear in the file. *(Note that the first three flags that appear in the file are not labeled.)*

Flag#1 (ibtyp) – It is always -2 and should not be changed.

Flag#2 (bvers) – Specifies the version number of the DPLR Code Package that contained the file template.

Flag#3 (ibdum) – Specifies if the file contains values for dummy cells. Allowable values are:

- 0 The file does not contain values for dummy cells.
- 1 The file does contain values for dummy cells.

nblk – Specifies the number of master blocks in the input grid.

idim – Specifies the dimensions of the simulation. Allowable values are:.

- 2 Two dimensional simulation
- 3 Three dimensional simulation

neq – Specifies the total number of coupled equations included in the matrix expression of the boundary conditions. The value is calculated as follows:

neq = ns+ner+nev+nee+idim+1

where: **ns** = number of chemical species in the flow
ner = number of rotational energy equations
nev =number of vibrational energy equations
nee = number of electron/electronic energy equations

net – Specifies the number of uncoupled turbulence equations.

nmc – Specifies whether there is a catalytic material map. Allowable values are:

0 No, a catalytic material map does not exist

1 Yes, a catalytic material map does exist

nme – Specifies whether a surface radiation map exists. Allowable values are:

0 No, a surface radiation map does not exist

1 Yes, a surface radiation map does exist

nmt – Specifies whether a transition map exists. Allowable values are:

0 No, a transition map does not exist

1 Yes, a transition map does exist

nmv – Specifies whether a view factor map exists. Allowable values are:

0 No, a view factor map does not exist

1 Yes, a view factor map does exist

Tech Tip: *A view factor map is used to account for the ability of concave block faces to "see" each other and thus describing how energy behaves within a concave surface geometry.*

f2 – Not used by DPLR at this time.

f3 – Not used by DPLR at this time.

f4 – Not used by DPLR at this time.

block sizes– Specifies the total number of computational cells in the *i, j, k* directions, respectively, for a master block.

Tech Tips:

1) *Although unlabeled, each line of values corresponds to one master block.*

2) *Values also found in the* `ntx`*,* `nty`*, and* `ntz` *flags in the block-specific areas of the DPLR input deck.*

`ibc numbers for each block -`Specifies the boundary condition values entered for each face of a master block in the DPLR input deck - `imin, imax, jmin, jmax, kmin, kmax` – respectively.

Tech Tip: *Although unlabeled, one line of values corresponds to one master block.*

Optional Lines

Catalytic material map specifiers, if `nmc=1`.

Surface radiation map specifiers, if `nme=1`.

Transition map specifiers, if `nmt=1`.

View Factor map specifiers, if `nmv=1`.

`Profile Data for Block #; Face # -` Listing of variable values specified by the `ibc` number for a particular master block face. For example, if Block #1, Face #6 has a value of `60,` DPLR will look at this file, in this place for numeric values for ρ_s, *u, v, w, Tv, T.*

Tech Tips:

1) *For boundary conditions to be accurately simulated, the data in this area must appear in the exact order the variable listing appears in the corresponding* `ibc` *flag entry*

.

2) *DPLR reads data in this file in the following order:*

1. *Any pointwise boundary condition numbers (first) and input profiles (second)*
2. *Any surface material maps*
3. *Any surface radiation maps*
4. *Any surface transition maps*
5. *Any surface view factor maps.*

3) *DPLR then loops in the following order:*
Inner loop over the face number (1-6), followed by a loop over the block number (1-nblk), then repeat the outer loop over read order listed above.

4) *Data should be written in standard "plot3d-like" format, in the order shown in the file above.*

6.4 Runtime Control Files

As you are monitoring a DPLR run, you may notice that your solution is not working the way you anticipated. In version 4.01.1 of DPLR Code Package, you can dynamically interact with a simulation mid-run to change timestep settings and DPLR's grid adaption values to correct problems without having to stop the run and start over through the use of a runtime control file.

A generic runtime control file named `generic.ctrl` is distributed with the DPLR Code Package Version 4.01.1 and can be found in the `cfdinput` directory.

6.4.1 Creating a Runtime Control File

Step 1: **Open** the text editor program for your system.

Action: At the command line prompt, type:

/[path to your cfdinput directory]/ `generic.ctrl`

Result: A generic input file appears on screen, with placeholder default values as shown below. To customize the file for your simulation, replace flags and values with those you want to change. Note that whenever a # sign appears in this file, DPLR considers whatever follows to be comments and will not parse the information.

Step 2: **Save** your control file to your working directory after giving it the same prefix as your solution file and adding the suffix `".ctrl"`.

DPLR will check for the existence of a control file in your working directory, by default, every 100 iterations during the solution run. If a control file exists and is correctly formatted, DPLR will read the new settings for the grid flags and/or timestepping flags and continue the simulation using those values.

```
# Run-time control example
nplot = 200;  iplot = 1
@iteration  500 ngiter=500 nalign=4, gmargin 2.5     # Trailing comment
@iteration 1000 igalign=1 nalign=3, gmargin 3.        # Another comment
@end
#####################################################################
# Iteration-number-dependent controls
#####################################################################
# istop
# nplot
# iplot
# nruntime_freq
#####################################################################
# Grid-tailoring controls
#####################################################################
# igalign
# ngiter
# nalign
# imedge
# imradial
# ngeom
# ismooth
# fs_scale
# ds_mult
# gmargin
# ds1
# cellRe
# ds1mx
# ds2fr
#####################################################################
# Miscellaneous controls
#####################################################################
# cfl
```

Figure 6-5 Example of Runtime Control File

6.4.2 Input Flags for Runtime Control Files

In DPLR Code Version 4.01.1, the following input flags can be dynamically changed in a runtime control file (see Section 4.2 for more information on each flag):

Global Flags: `istop, nplot, iplot, nruntime_freq`

Tech Tip: *`nruntime_freq` is not found in the standard DPLR input deck. It is used to specify how often the control file is to be read by DPLR, i.e., after n iterations. Default value = 100.*

Grid-Adaption Flags: `igalign, ngiter, nalign, imedge, imradial, ngeom, ismooth, fs_scale, ds_mult, gmargin, ds1, cellRe, ds1mx, ds2fr`

Timestepping Flags: `cfl`

Tech Tip: *Note that changes to the `cfl` number list should be made via an iteration-specific command as illustrated below.*

6.4.3 Syntax for Runtime Control Files

Unlike other DPLR file formats, you do not have to use any of the control file input flags in any particular order. Also, the syntax for this type of input file is as follows:

- **For comments**, type a # sign first, and everything after that on that line will not be "seen" by the code. For example:

  ```
  #   I am expanding this grid.
  ```

  would not cause DPLR to change the running simulation in any way.

- **For a generic command**, type the name of the flag you want to change followed by an = sign followed by a numeric value. More than one generic command can appear on a line, but they should be separated by a comma or colon or semicolon. For example:

  ```
  nplot=300;  iplot=2
  ```

  would result in DPLR changing the value of those flags when it reads the control file after 100 more iterations.

- **For an iteration-specific command**, type an @ sign, followed by the word 'iteration' , followed by an = sign, followed by a numeric value. Then type the

name of the flag you want to change at that iteration, an "=", then a numeric value. An iteration-specific command can also have several flags associated with it, but they should be separated by a comma, colon or semicolon. For example:

```
@iteration 2000 igalign=3; imradial=2
```

would result in DPLR changing the value of those flags when it reaches iteration 2000, assuming that iteration had not already been passed when the control file was read.

When changing a `cfl` number listing, the following syntax should be used:

```
@iteration 2000 cfl = 100
@iteration 2100 cfl = 250
@iteration 2500 cfl = 500
```

This entry in a control file in your working directory will tell DPLR that when it reaches the first iteration in this list that has not yet been passed, it will adopt this timestepping schedule in place of the one in the DPLR input deck and not refer back to that original listing unless the run is stopped and restarted.

6.5 Restart Files

The restart file is the DPLR solution file. It is named via `fname` in the DPLR Input Deck and typically has the suffix `"*.pslx".`

The first time a simulation is run, DPLR writes a restart file as frequently as specified in the `nplot` input flag and saves as many restart files as specified in the `iplot` input flag.

Restart files contain all the input deck values and physical modeling parameters that were used in the simulation. Once written, a restart file is linked within DPLR to the binary, machine-readable "*.pgrx" grid file that was used for the simulation.

Tech Tip: *Although restart files can be written in unformatted parallel* `("*.psln")` *and ASCII parallel* `("*.psla")` *formats, the preferred format in the DPLR working environment is XDR parallel* `("*.pslx")` *- a binary, machine-readable file.*

6.5.1 Converting Function Files to Restart Files

The only CFD solution file (aka "function" file) that can be used as direct input to DPLR is a `"*.pslx"` restart file. Thus, if you want to rerun a solution in DPLR that

was originally created in another CFD solver application such as ***radial_interp*** or ***SAGe***, you must first convert the function file from that program (in these cases a plot3d file) into a "*.pslx" restart file – a task that can be accomplished using FCONVERT.

Step 1: **Make sure:**

1) the input function file contains the following variables in the following order:

$$\rho_s, u, v, (w), T, (T_R), (T_V), (T_e), (turb)$$

where r_s are the species densities, u, v, and w are the velocity components, T is the translational temperature, T_r is the rotational temperature, T_v is the vibrational temperature, T_{el} is the free electron temperature, and *turb* are the turbulence variables.

2) the input function file has dimensions of the number of internal cells in each grid block if `idummy` = 0, or the number of internal cells + 2 to account for a single row of dummy cells if `idummy` = 1.

Tech Tip: *The second layer of dummy cells, used for high order flux extrapolations, should never be included in the input function file. Either style can be used to create restart files. If dummy cell information is not provided in the function file, values in the dummy cells will be extrapolated from the interior, and then overwritten by the corrected values when DPLR is run. If the dummy cells are included in the file, the values contained in the dummy cells should be face centered values at all solid surfaces. This allows for an exact specification of the viscous wall boundary condition. If dummy cells are not included, the boundary condition at any viscous walls will be reinitialized on restart, which will lead to a significant perturbation to the flowfield and L2norm residual*

Step 2: **Open** an FCONVERT input deck file (See Section 3.1) and set `iaction=10, ifile=2, inform=3 or 23`, `nsin` = # of chemical species, `nerin, nevin, necin` = # of independent temperatures in each mode, `ntbin` = # of turbulence variables, and the rest of the flags to problem-specific values.

Step 3: **Save** the input deck file.

Step 4: **Run** FCONVERT.

Tech Tip: *During the conversion process, FCONVERT will generate all necessary header elements and format the file properly for DPLR. However, the resulting restart file does not contain all of the CFD modeling flags, and thus cannot be post-processed with POSTFLOW until it has been run at least 1 iteration and re-saved in DPLR.*

6.6 Chemistry Files

The DPLR Code Package contains a large number of chemistry model files that are automatically placed in the `cfdinput` directory when the software is installed along with other physical property databases. (See Section 2.4 for more information on the contents of directories installed with the DPLR Code Package Version 4.01.1).

Chemistry files contain the input information DPLR needs to define species lists, chemical kinetic reactions, and reaction rates for a simulation. A chemistry file is required input for all simulations and should be specified to DPLR by the `cname` variable in the DPLR input deck along with an absolute pathname.

To help you choose the chemistry file that is most appropriate for your simulation, file names typically contain a descriptive indication of the flow environment being modeled, the number of chemical species included in the model, the personal or institutional source of the model and the year the model was published, followed by a "*.chem" suffix. For example, the file name:

```
air7sp-park93.chem
```

tells you that it is a model of earth "air" containing seven chemical species, that it was developed by Park, and that it was published in 1993.

Tech Tip: *For a more complete description of the model contents, reference publication, and author(s), see the legend at the end of each "*.chem" file.*

6.7 Radiation Files

The radiation coupling file is an optional input file used to input pointwise $\nabla \cdot Q_R$ information obtained from an offline radiation transport code. This plot3D-formatted file name typically has a "`*.pdrx`" suffix and is specified by the `rname` variable in the DPLR input deck.

An example of a radiation file for a 2 block grid where one block size is 32x64x64 points and a second block size is 64x32x64 points is given below:

```
2
32    64    64    1
64    32    64    1

[block radiation data in the i, j, and k
directions]
```

The first integer in the first line is the number of blocks. The four integers in the next line is the number of points in the first block in the i,j,k directions followed by a "1" for one variable, then repeated for the points in the second block on the next line. This is followed by the real block radiation data supplied by the offline radiation transport code.

Tech Tip: *Although optional, if an "`rname`" is specified in the DPLR input deck, a "`*.pdrx`" file must exist in the `cfdinput` directory to avoid a runtime error.*

6.8 Convergence Files

When `ires >0` in the DPLR input deck, DPLR automatically creates a convergence file when a simulation is run and places it in your working directory.

The convergence file contains information on the iteration number, CFL number or timestep, and L2norm of the flow variable specified by the `iresv` flag of the DPLR input deck.

An example of a convergence file where `ires=2` and `iresv=1` is given below.

```
# Summary of enabled CPP compiler directives:
# --> AMBIPOLAR=1,PARKTEXP=0.50,SCEI=1.00,NOHTC


# computing L2norm residual of density
   # nit     global resid            cfl
      1   1.000000000000000E+00   1.0E-03
      2   9.999989632126156E-01   1.0E-03
      3   9.999980913475810E-01   1.0E-03
      4   9.999975461771219E-01   1.0E-03
      5   9.999973229450715E-01   1.0E-03
                  ..........
                  ..........   etc.
     98   8.653047974124419E-01   2.5E+00
     99   8.637292011393368E-01   2.5E+00
    100   8.621722044117890E-01   2.5E+00


# Loop time =  8.75 seconds on  8 processors
```

Convergence files, named with the same prefix as the restart file and the suffix "*.con", are usually retained for archival purposes and can be used to plot the rate of convergence of a specified variable in a given simulation.

Tech Tip: *If the current job began with a restart file, DPLR appends the new convergence data to the existing file (if any). If the current job is a fresh start, a new file is created and any previous file with the same name is automatically overwritten.*

6.9 Aerodynamic Files

When `ires=5` or `15` in the DPLR input deck, DPLR automatically creates an aerodynamic datafile when a simulation is run and places it in your working directory.

The aerodynamic file contains information on the iteration number and the three force and moment coefficients computed as dimensional quantities. Moments are computed about the origin (0, 0, 0), and vehicle symmetries are not incorporated.

An example of an aerodynamic file where `ires=5` is given below.

```
#   nit     Fx             Fy          Fz           Mx            My           Mz
    2001    2.5777E+05     4.706E+04   8.5400E-02 -1.0967E-01    -6.2569E-02 2.7165E+05
    2002    2.5777E+05     4.706E+04   8.5075E-02 -1.0972E-01    -5.8149E-02 2.7165E+05
    2003    2.5777E+05     4.706E+04   8.4872E-02 -1.0975E-01    -5.4368E-02 2.7165E+05

…… etc.
```

Aerodynamic files, named with the same prefix as the restart file and the suffix "*.aero". Typically used as an additional means of monitoring the progress of a simulation run toward convergence, these files are usually retained for archival purposes and can be used to plot the rate of convergence of a given aerodynamic simulation.

6.10 Log Files

DPLR automatically creates a log file or standard out when a simulation is run and places it in your working directory. Log files, names with the same prefix as the restart file and the suffix `"*.log",` are usually retained for archival purposes and

can be parsed to automatically fill out quality check forms if those are used as part of your CFD process.

The log file contains a subset of the same information that is echoed to your screen in a standard out (STDOUT).

An example of a log file is given below.

```
# Summary of enabled CPP compiler directives:
# --> AMBIPOLAR =  1
# --> PARKTEXP = 0.50
# --> NOHTC

# Air Mechanism: 5 species, 5 reactions (Park 1990) Model
# --> Species List: N2 O2 NO N O
# --> Reaction rates from: air5sp_park90.chem
# --> Reaction Status: 1 1 1 1 1
# --> Keq Fit Used   : 0 0 0 0 0
# --> NASA Lewis thermo fits used to find Keq
# --> Assume molecules created/destroyed at mixture Tve

# Catalytic wall BC enabled
# --> Constant accomadation coeff;  gamma = 1.000

# Rotational Equilibrium - Fully Excited

# Vibrational Non-Equilibrium - SHO

# Electronic Energy Neglected

# Laminar Navier-Stokes Simulation
# --> Gupta-Style Collision Integrals & Yos Mixing Rule
# --> Taking gradients of ev
# --> Multi-Species Binary Diffusion (Mole Fraction Gradients)
# --> Binary diffusion coefficients from Gupta Collision Integrals
```

```
# --> SCEBD model used to compute diffusive fluxes

# Ideal Gas Equation of State

# Axisymmetric Flow - rotate about x-axis

# Implicit - Data Parallel Line Relaxation; kmax =  4
# --> Using Global Timestepping

# INFORM: saving  2 previous restarts



# Freestream Reynolds    Number = 8.260E+06 (1/m)
# Freestream Frozen Mach Number = 1.087E+00
# Freestream Equil. Mach Number = 1.133E+00
```

Figure 6-6 Example of Log File

Tech Tip: *To avoid having DPLR overwrite archival output files with files generated by re-runs of simulation, rename your restart file before beginning each run so that all the automatically created output files will indicate which simulation run created them.*

6.11 Tecplot Files

Amtec's Tecplot® visualization software is a tool often used to process results of DPLR simulations. For this reason, POSTFLOW has the capability of writing dataset files in two Tecplot-specific formats:

- Tecplot binary (`"*.plt"`)
- Tecplot `ASCII ("*.dat")`

As noted in Section 2.2, 5.2, and 5.4, however, the Amtec-provided "tecio.a" (or "tecio64.a") runtime library must be installed on your system to generate binary "*.plt" files and may be available from the Tecplot website at:

http://www.tecplot.com.

Chapter 7 - DPLR Workflow

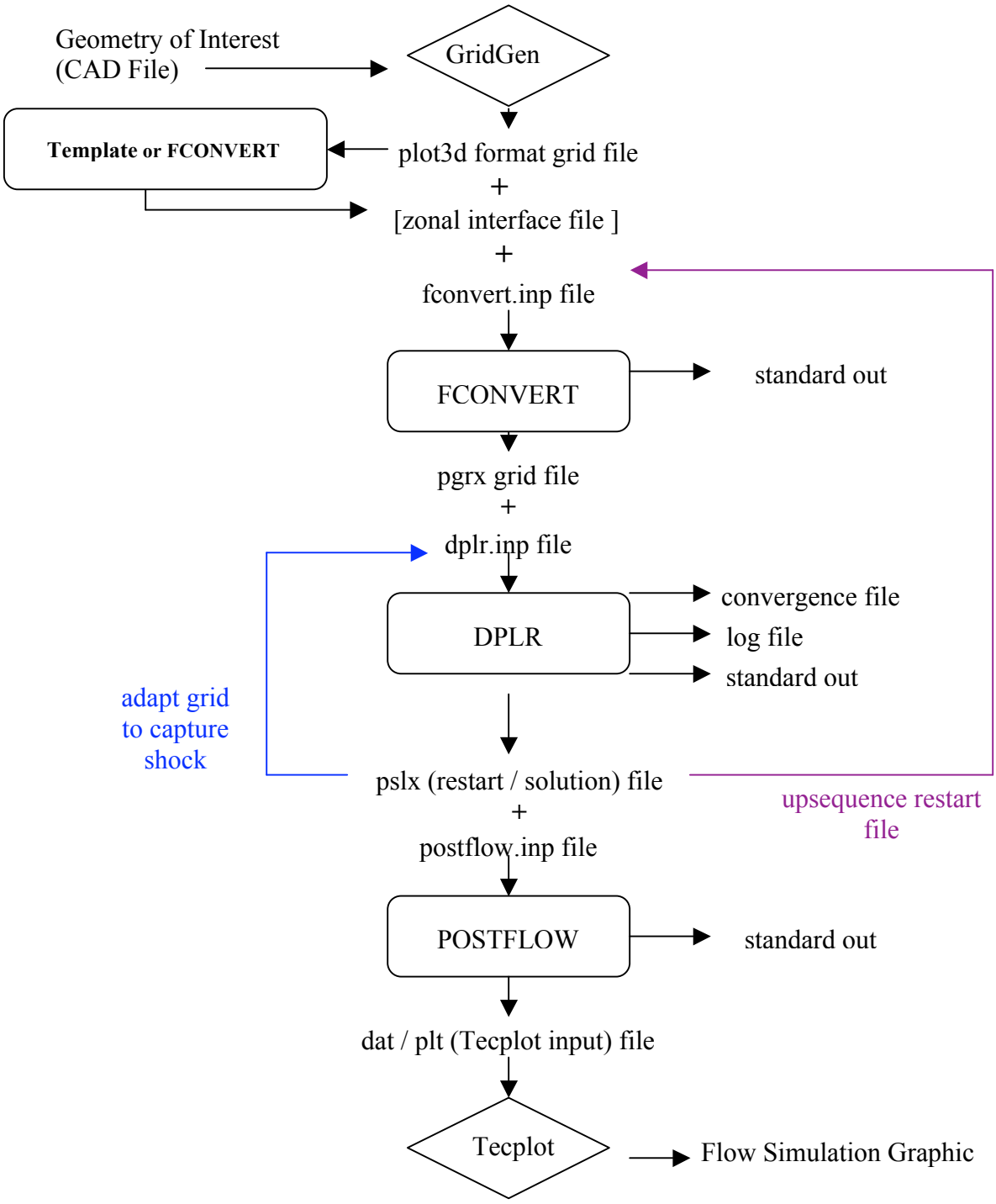
Contents

7.0 Introduction

Using the DPLR Code Package to achieve hypersonic flow simulation solutions can be a complex undertaking. Although the main tasks, i.e., grid file generation, grid file conversion, solution processing, and data extraction, are essentially sequential in nature, they are also iterative and often require concurrent execution to make the most productive use of your time, your tool, and your computing resources.

This chapter will suggest a set of actions or "flow of work" that may help you achieve solutions in a more timely manner. Once you become familiar with the use and robust capabilities of the DPLR Code Package, however, it is likely that you will develop your own customized workflow.

7.1 DPLR Work Flow Chart



7.1.1 Initial Simulation Run

Geometry of Interest ⟶ GridGen ⟶ plot3d grid file

This first time you run a flow simulation problem for a geometry of interest (most likely a vehicle of some sort) using the DPLR Code Version 4.01.1 Package, you will need to generate a new (or adapt an existing) structured computational grid that you believe will capture the shock wave the object will encounter at hypersonic speeds within a specified flow environment. This "first guess" can be created from specifications in a CAD design or from scratch. In either case, you will need to use a grid generation program such as GridGen or GridPro to develop the grid file for use with DPLR. In most cases, the preferred form of the structured grid file you create will be `plot3d.`

With DPLR Code Version 4.01.1, you have the option of processing your `plot3d` grid file through SUGGAR to enable overset grid capabilities with your simulation, assuming you have successfully installed the required third-party software (SUGGAR) and data libraries (DirTlib) before compiling the DPLR Code Package as discussed in Chapter 8. After processing with SUGGAR, you will have an overset-capable version of your `plot3d` grid file which may (or may not) have an altered block order along with a domain connectivity file (.dci) in your home directory.

plot3d grid file + fconvert.inp ⟶ FCONVERT ⟶ pgrx grid file, standard out

Once you have created your structured grid file, you will need to create an input file for FCONVERT that specifies, among other things, how many processors will be used to run your simulation and how your grid file should be "decomposed" for parallel processing. When both files are available, you will run the DPLR file conversion executable, FCONVERT, to create a structured grid file that can be read by DPLR. In most cases, the recommended form of the structured grid file you create through this file conversion process will be `pgrx`. In addition to the new grid file, FCONVERT will create a screen report called a "standard out" of the actions taken to create the pgrx file. This report file can be saved for archival purposes.

pgrx grid file + dplr.inp ⟶ DPLR ⟶ pslx file, log file, convergence file, standard out

When the pgrx file for your problem is prepared, you will then create an input file for DPLR that specifies a variety of information about the flow environment your object will encounter, conditions of flow entry, conditions at the surface of different portions of your object, and the timestepping regimen you want DPLR to employ during its solution calculations. When both files are available, you will run the appropriate

DPLR executable, DPLR2D or DPLR3D, to create a solution or restart file. In most cases, the recommended form of the solution file will be `pslx`. In addition to the restart file, DPLR will also create a log file, a convergence file, and another standard out file to capture various aspects of the progress of the solution run. All three of these report files can be saved for archival purposes.

pslx file + postflow.inp ⟶ POSTFLOW⟶ flow.plt, standard out

When DPLR completes the specified number of iterations to achieve a solution and write a restart file, you will need to create an input file for POSTFLOW that specifies the data you want extract from the solution and the format of the file you want POSTFLOW to write – something that will depend upon the third-party data reporting or visualization program, such as Tecplot, you are using. When both files are available, you will run the DPLR data extraction executable POSTFLOW, to create an input file for your flow simulation graphics program. If you are using Tecplot, the form of this file will be `.plt.` In addition to the new input file, POSTFLOW will create a screen report called a "standard out" of the actions taken to create the file which, again, can be saved for archival purposes.

flow.plt → Tecplot → graphic representation of simulation solution

When the post-process data file is available from POSTFLOW, you can launch your data visualization program to read in the information and create a graphic representation of your simulation run.

7.1.2 Subsequent Simulation Runs

Although you began your solution run with a structured grid that represented your "best guess" for capturing the shock wave in a hypersonic flow simulation problem, it is common to find that some adjustment of this grid is needed for your solution to adequately converge. These adjustments can be accomplished in subsequent runs of your simulations using the following technique.

pslx file + revised DPLR.inp⟶ DPLR → revised pslx file, log file, convergence file, standard out

After DPLR has created a restart file (pslx) that represents a solution needing some "adjustments", create a revised DPLR input file by creating and renaming a copy of the dplr.inp file you used for your initial run. In your new input file, enter the following input flag settings: iinit=1 ; igalign=1; nalign=4; ngiter=500.

These new settings tell DPLR that the run will be using a restart file, that grid alignment is to take place, that four such alignments will take place during the simulation, and that they will occur every 500 iterations of the run.

When the revised pslx file is created and data is extracted by POSTFLOW and visualized by a graphics program, you can decide if further "adjustment" to your structured grid is needed to capture the shock wave. If so, repeat the process above, but consider setting a more aggressive CFL ramping schedule (i.e., greater time stepping intervals) as you approach a converged solution.

7.2 Workflow Shortcuts

Over the years, a variety of tools and procedures have been developed to decrease the time spent in creating and running DPLR simulations. This section describes several of the more commonly used of these workflow shortcuts.

7.2.1 Sequence the Grid

As discussed in Section 3.5, computational grids composed of a large number of data points typically take longer to solve than grids with fewer points. As a result, grids used for initial solutions of CFD problems are sometimes coarsened or "sequenced" to reduce the number of points while maintaining the topology of the mesh. After an acceptable "first guess" is acquired, the grid is restored in a step-wise fashion to its original number of points for final solution and post-solution data reporting.

To sequence or coarsen an input grid, open the fconvert.inp file, and enter the following settings: imseq=1; iseq=*n*; jseq=*n*; kseq=*n* where *n* is the number of times the grid for that block should be coarsened in the *i, j, k* directions. (See Section 3.5.1 for more information on this technique).

To restore grid points and refine your solution, use FCONVERT to upsequence your restart file and create a new pgrx file that matches the refined level of your upsequenced restart file.

"coarsened" restart file + 1st revised fconvert.inp → FCONVERT → upsequenced restart file

plot3d grid file + 2nd revised fconvert.inp → FCONVERT → refined pgrx grid file that matches points in the upsequenced restart file

This can be accomplished by following the steps below:

Step 1: **Open and name** a new fconvert.inp file**.**

Step 2: **Set** `ifile=2, inform=11, imseq=-2, iseq, jseq, kseq to values used during sequencing, iname= coarsened restart file name, oname= new (less sequenced) restart file name (*.pslx).`

Step 3: **Save** file to your working directory.

Step 4: **Run** FCONVERT < new FCONVERT input file.

Step 5: **Open and name** another new FCONVERT input file.

Step 6: **Set** `ifile=1, inform=2, imseq=0, iname= original plot3d grid filename, oname= new (less sequenced) XDR parallel grid file name (*.pgrx).`

Step 7: **Save** file to your working directory.

Step 8: **Run** FCONVERT < second new FCONVERT input file.

Your working directory now contains an upsequenced restart file that can be used to start a new solution run along with the DPLR-readable grid file containing the same number of data points as the upsequenced solution file. See Section 3.5.2 for more information on Mesh Sequencing.

7.2.2 Use Runtime Control Files to Adjust Grids and CFL Schedules

With DPLR Code Version 4.01.1, you no longer need to wait until your initial solution run is complete to adjust your "first guess" grid or change your CFL timestepping schedule, and then re-run the simulation. Using a runtime control file, you can dynamically interact with a simulation mid-run while monitoring the progress of convergence with concurrently running graphic visualizations of restart files as they are being written during your DPLR run.

By using this option, you may avoid the need to repetitively stop and restart simulation runs. (See Section 6.4 for more information on creating and managing Runtime Control Files).

7.2.3 Use ***Template*** To Create DPLR Input and Zonal Interface Files

Manually creating DPLR input and zonal interface files can be a time-consuming task. However, by using the ***Template*** utility, created by Scott Thomas and David Saunders and distributed with the DPLR Code Package Version 4.01.1, these two tasks can be automated.

plot 3d grid file + sample.inputs file → TEMPLATE → dplr.interfaces, dplr.inputs, gasp.inp, template.con

To use ***Template*** to automatically create zonal interface files and block-specific areas of the DPLR input file, perform the following steps:

Step 1: **Rename** the 'generic.inp' file in the cfdinput directory as 'sample.inputs' and save it to your working directory.

Step 2: **Place** the structured plot3d grid file of your object of interest in your working directory.

Step 3: **Run** ***Template***.

Your working directory now contains four new files: dplr.inputs; dplr.interfaces; gasp.inputs, template.con.

When you open the dplr.inputs file, you will see that ***Template*** has created content for the block-specific areas of your DPLR input file. You may use this content as a guide to enter the values manually or simply copy and paste it into the DPLR input file you are creating for your simulation run.

When you open the dplr.interfaces file, you will see that ***Template*** has created a zonal interface file for use in your simulation. This method detects full-face interfaces only, unlike FCONVERT which has the option of detecting subfacing through different settings of `inint`. Thus, the zonal interface file generated by ***Template*** should only be used when no sub-face interfaces exist in the computational grid.

See Section 9.1.5 for a more complete discussion of the ***Template*** utility.

7.2.4 Understand Your Computing Resources

The efficiency of the DPLR Code Package as a CFD solver depends, in part, on the number of processors available for parallel solution of your flow problem. Thus, the

more processors you can allocate to your simulation run, the less time it will take to achieve a solution.

In addition to raw computing power, however, knowing the exact number of processors that can be dedicated to your solution will allow you decompose the plot3d input grid into computational blocks that can be most efficiently handled by your computing resources. This is accomplished in the FCONVERT input file by setting `iaction=2; nbreak=`*n* where *n* is the number of available processors.

Chapter 8 - Using Overset Grids

Contents

8.0 Introduction

An overset grid capability has been added to the DPLR Code Package beginning with Version 4.01.1. Overset grid techniques can extend traditional structured grid approaches to problems with greater geometric complexity and can be used to facilitate automated grid-generation or rapid analysis of configurations during design.

This chapter will describe modifications to the installation process required to enable the overset grid capability in the DPLR Code Package as well as modifications to the typical DPLR workflow that arise due to the use of overset grids.

8.1 Installation

This section details special installation steps required to make the overset logic available within the DPLR Code Package, as well as the steps necessary to compile several utilities that may be useful for pre- and post-processing.

8.1.1 Installation of the DPLR Code Package

The overset capability in DPLR is built upon the DiRT and P3D libraries [1], which must be compiled separately and included in the link step for the DPLR Code Package executables. Each makefile in DPLR has been modified to access these libraries via environment variables, DIRTLIB_DIR and DIRTINC_DIR for the DiRT library (DiRTlib) and P3DLIB_DIR for the accompanying Plot3D library (P3Dlib). Therefore, for the overset grid capability to be available in DPLR, these two libraries must be compiled, and the assoicated environment variables must be defined.[1]

Step 1: **Define** the environment variables.

Action: Depending on your environment, include in your login script (e.g. `.cshrc` file) commands such as the following:

```
setenv DIRT_HOME $(HOME)/src/DiRTlibV1.36/src

setenv DIRTINC_DIR $(DIRT_HOME)

setenv DIRTLIB_DIR $(DIRT_HOME)
```

[1] DPLR Version 4.01.1 requires DiRTlib version 1.36 or greater.

```
setenv P3D_HOME $(HOME)/src/P3Dlib/src

setenv P3DLIB_DIR $(P3D_HOME)
```

Result: `DIRTINC_DIR` will specify the directory containing the drt_version.h header file, `DIRTLIB_DIR` will specify the directory containing the DiRT library file, and `P3DLIB_DIR` will specify the directory containing the P3D library file (see below).

Step 2: **Compile** the P3D library.

Action: At the command line prompt, type the following commands:

```
cd $P3D_HOME

rm *.o *.a

make

mv libp3d.a $P3DLIB_DIR
```

Result: Assuming there were no problems, the archive file `libp3d.a` is created and stored in the directory specified by the `P3DLIB_DIR` environment variable. Also created is the executable file `p3dconvert`, which will be used later in this chapter.

Step 3: **Compile** the serial version of the DiRT library.

Action: At the command line prompt, type the following commands:

```
cd $DIRT_HOME

make serial

mv libdirt.a $DIRTLIB_DIR

mv drt_version.h $DIRTINC_DIR
```

Result: Assuming there were no problems, the archive file `libdirt.a` is created and stored in the directory specified by the `DIRTLIB_DIR` environment variable.

Step 4: **Compile** the parallel version of the DiRT library.

Action: At the command line prompt, type the following commands:

```
cd $DIRT_HOME

make mpich

mv libdirt_mpich.a $DIRTLIB_DIR
```

Result: Assuming there were no problems, the archive file `libdirt_mpich.a` is created and stored in the directory specified by the `DIRTLIB_DIR` environment variable.

Tech Tip: *The compilation command for the parallel version of the DiRT library may vary considerably from platform to platform. On systems using mpicc, the build process for the current step may be simplified considerably by defining the environment variable MPICH_ROOT to specify the directory in which bin/mpicc is located (such that $MPICH_ROOT/bin/mpicc specifies the full path). On other systems, the mpich target in $DIRT_HOME/Makefile will need to be edited appropriately. Refer to the $DIRT_HOME/README file for further instruction.*

Step 5: **Compile** the DPLR Code Package. (See Section 2.3 for instructions on installing the baseline DPLR Code Package.) Ensure that the `DIRTINC_DIR`, `DIRTLIB_DIR` and `P3DLIB_DIR` environment variables are defined as described in Step 1 above. Then compile (or recompile) the DPLR Code Package.

Action: At the command line prompt, type:

```
cd $DPLR_HOME

make clean

make
```

Result: Assuming there were no problems, all executable files in the package are created. Links to executables `dplr2d`, `dplr3d`, `fconvert`, and `postflow` are located in the `bin` directory.

Tech Tip: *The DPLR* `makefile.comm` *file includes logic to define the OVERSET CPP flag as part of the CPPFLAGS variable when the environment variables described in Step 1 above are defined. If OVERSET is not defined, most if not all of the overset logic is stripped from the code prior to creation of the object files. As a result, a make clean on the entire DPLR Code Package is required when switching between non-overset and overset compilations.*

Step 6: **Compile** additional overset utilities.

Action: At the command line prompt, type:

```
cd $DPLR_HOME/utilities/overset

make
```

Result: Assuming there were no problems, all executable files in the overset utilities directory are created. Examples include `merge_dplr` and `merge_usurp`, which will be used later in this chapter. All of the utilities are listed and briefly described in Section 8.2.

Tech Tips:
1)*. The Makefile in $DPLR_HOME/utilities/overset specifies the Fortran 90 compiler and compiler flags. These can be modified directly or on the command line by specifying the FORT and FFLAGS variables.*

2). *Adding $DPLR_HOME/utilities/overset to your path is recommended in order to simplify access to the executables in that directory.*

8.1.2 Installation of USURP

USURP is a separately packaged utility that allows for accurate surface integration in grid systems that contain overlapping surface meshes [2]. It is written in Fortran 90/95 and thus requires a working f90 compiler on the destination machine. USURP also utilizes some third-party routines written in C.

USURP is distributed as a gzipped tar file named for the specific version of the code and date of release; e.g. `usurp_v244_02242009.tgz`. Version 2.44 of USURP is the first that includes support for DPLR Version 4.01.0 input files.

Step 1: **Define** the environment variables. In order to read the preferred FXDR-formatted DPLR grid files, USURP must be linked with FXDR at compile time. For this to occur, the environment variable `FXDR_HOME` must be defined to specify the full path of the FXDR library (including the name of the library file itself, e.g. `libfxdr.a`).

Action: Depending on your environment, include in your login script (e.g. `.cshrc` file) commands such as the following:

```
setenv FXDR_HOME $HOME/src/fxdr_2.1c/libfxdr.a
```

Result: FXDR_HOME will specify the full path of the filename of the FXDR library.

Step 2: **Unpack** the USURP files.

Action: At the command line prompt, type:

```
tar xvzf usurp_v244_02242009.tgz
```

Result: A directory structure is created with the new directory `SOURCE2.44` as the root.

Step 3: **Print** the usage for the USURP make file.

Action: At the command line prompt, type:

```
cd SOURCE2.44/src

make
```

Result: The USURP `Makefile` prints a list of targets that can be used to compile the executable, drawing from the `Make.sys` file.

Step 4: **Create** the executable file.

Action: Select an option from the result of Step 3 and type the respective command. For example, in an environment that contains the Intel Fortran 90 compiler `ifort`, type the following at the command line prompt:

```
make intel8_little
```

Result: Assuming there were no problems, the executable file `usurp` is created.

8.2 Utilities

The following codes or scripts are provided with the DPLR Code Package in the utilities/overset directory:

- calc_dirt_ijk
- convert_to_2d
- dplr_grid_to_suggar
- gg2dplr
- gg2suggar
- interrogate_dplr
- merge_dplr
- merge_usurp
- overflowdnamelist2xml
- peg2xml
- plot_suggar_2d
- plot_suggar_3d

- run_suggar_2d
- run_suggar_3d
- scan_orphans
- update_suggar_2d
- update_suggar_3d

calc_dirt_ijk – a Fortran 90 code that converts i,j,k indices from DPLR into a 1D index used by SUGGAR or vice versa. Input is interactive in response to prompts.

convert_to_2d - a Fortran 90 code that can be used to convert 2D grids from an unformatted multiblock PLOT3D file containing ni, nj, and nk values in the header (where nk=1) to an ASCII 2D PLOT3D format containing only ni and nj values in the header. (The latter format may be more to the liking of FCONVERT.) Input is interactive in response to prompts.

dplr_grid_to_suggar – a Fortran 90 code that can be used to convert 2D grids from an unformatted multiblock PLOT3D file containing only ni and nj values in the header to an unformatted multiblock PLOT3D file containing ni, nj, and nk values in the header (where nk=1). (The latter format is required by SUGGAR.) Input is interactive in response to prompts.

gg2dplr - a rudimentary Fortran 90 code that uses a generic Gridgen boundary condition file to generate a baseline DPLR input file. Input is interactive in response to prompts.

gg2suggar - a Fortran 90 code that uses a generic Gridgen boundary condition file to generate a baseline SUGGAR input file. Input is interactive in response to prompts.

interrogate_dplr - a Fortran 90 code that allows the user to manually interrogate the values written by POSTFLOW to a cell-centered Tecplot file. The output format in POSTFLOW should be specified with `ouform=25` and `interp=11`, with `ivarp` consisting of any desired dependent variables. Input is interactive in response to prompts.

merge_dplr - a Fortran 90 code that combines vertex-based grid coordinates with cell-centered values of iblank (obtained from the DCI file) and any primitive variables output from POSTFLOW. See Section 8.6 for more details.

merge_usurp - a Fortran 90 code that combines output data from POSTFLOW with a unique surface definition output from USURP (i.e. having removed any overlapping regions from the surface mesh). See Section 8.6 for more details.

overflowdnamelist2xml - a Fortran 90 code that uses an OVERFLOW-D input file (redirected from stdin) to generate a baseline SUGGAR input file (output to stdout). Usage:

```
overflowdnamelist2xml < [input namelist]
```

peg2xml - a Fortran 90 code that uses a PEGASUS input file (redirected from stdin) to generate a baseline SUGGAR input file (output to stdout). Usage:

```
peg2xml < [input namelist]
```

plot_suggar_2d - a Fortran 90 code that converts the results from SUGGAR (the composite grid and iblank information) into a Tecplot data file. Input is interactive in response to prompts.

plot_suggar_3d - a Fortran 90 code that converts the results from SUGGAR (the composite grid and iblank information) into a Tecplot data file. Input is interactive in response to prompts.

run_suggar_2d - a shell script containing the commands typically needed to run SUGGAR for a 2D case. The script assumes that the `suggar_2d.linux` executable is in the user's path.

run_suggar_3d - a shell script containing the commands typically needed to run SUGGAR for a 3D case. The script assumes that the `suggar_3d_opt.linux` and `surfasm` executables are in the user's path.

scan_orphans - a Fortran 90 code that writes out regions of orphans from a SUGGAR DCI file.

update_suggar_2d - a shell script containing the commands typically needed for grid adaption in a 2D case. In order to be used by DPLR, it must be copied to a file named `update_suggar` and placed in the main DPLR working directory.

update_suggar_3d - a shell script containing the commands typically needed for grid adaption in a 3D case. In order to be used by DPLR, it must be copied to a file named `update_suggar` and placed in the main DPLR working directory.

8.3 Pre-Processing

The primary change brought about by the use of overset grids is in the grid generation step and in the necessity to generate the domain connectivity information, or DCI, for the overset assembly. Both of these subjects are outside of the scope of the current document beyond a few brief remarks.

The overset grid generation methods for DPLR are in theory no different than for any other code, and in that respect, the usual grid generation packages such as Gridgen/Pointwise and Chimera Grid Tools may be applied. It is recommended that users who are new to overset grids refer to the paper "Best Practices in Overset Grid Generation" [3] for a valuable introduction to the topic. Users must bear in mind that cell-centered solvers such as DPLR will require more overlap than vertex-based solvers such as OVERFLOW, and this must be accounted for when the grids are first created.

The domain connectivity information is generally comprised of an iblank array which designates each cell in the grid as either IN, OUT, FRINGE, or ORPHAN, along with the interpolation stencil and interpolation weights associated with each FRINGE cell that specify how data is communicated at overset boundaries. This information is normally generated by grid assembly software, of which there are many options.

NASA has long been a lead organization for the development of overset methods and supports a wide array of overset tools popular in government and industry. Among these is Pegasus, a grid assembly tool that targets vertex-based flow solvers such as OVERFLOW.

Compared to OVERFLOW, DPLR simulations are characterized by several features that require special attention during the overset assembly process. For example, DPLR generally utilizes a cell-centered, full-viscous stencil on grids that may include point-matched, block-to-block interfaces. As of this writing, the only overset grid assembly tool known by the current authors to explicitly support all of these requirements is SUGGAR [4], which is therefore the recommended grid assembly tool for DPLR.

8.3.1 Special Considerations for SUGGAR

Detailed instructions for running SUGGAR are beyond the scope of the current document beyond a few brief remarks regarding the grid specification and SUGGAR input file.

For DPLR users, the easiest way to import grids into SUGGAR is using the PLOT3D format. Gridgen users, for example, should export the grid as a double-precision,

unformatted, PLOT3D volume grid. (Note that a volume grid should be created and exported even for 2D grids.)

SUGGAR currently requires that any PLOT3D-formatted input grid files be separated into single-block grid files. The p3dconvert utility included with P3Dlib provides a simple mechanism to split an existing multiblock grid file using the command

```
p3dconvert [input name] -sp3dudl [output name]
```

As an example, specifying an output name of Grids/block.grd results in a series of files with the names `Grids/block_1.grd, Grids/block_2.grd,` etc.

Several utilities have been provided in the $DPLR_HOME/utilities/overset directory to facilitate the creation of a baseline SUGGAR input file. Gridgen users, for example, should export the boundary conditions from Gridgen using the generic analyis software (AS/W) format and convert the resulting file to a SUGGAR Input.xml file format using the provided gg2suggar utility. Users of Pegasus or OVERFLOW may want to investigate the provided peg2xml or overflowdnamelist2xml utilities. Users of Chimera Grid Tools may be able to export a SUGGAR input file directly using a Tcl script written for CGT.

The following settings in the SUGGAR input file are recommended for DPLR users:

```
<cell_centered marking_using_neighbors="y"/>

<fringe_stencil type="diag+planar_first_offdiag"/>
```

If the thin-layer approximation to the viscous terms is applied in DPLR, then the `fringe_stencil` element may be omitted.

Run SUGGAR using one of provided utility scripts or the appropriate command line syntax. For 2D cases, the provided run_suggar_2d script is recommended, or the user may execute directly from the command line using a command such as:

```
suggar_2d.linux Input/Input.xml
```

For 3D cases, the provided `run_suggar_3d` script is recommended, or the user may execute directly from the command line using a command such as

```
suggar_3d_opt.linux -run_surfasm "" Input/Input.xml
```

8.3.2 Special Considerations for FCONVERT

Generally speaking, no modifications are necessary to the steps normally taken with FCONVERT during DPLR set-up. When using FCONVERT to convert the grid format or split the grid for parallel processing, there is no need to specify in any way that the grid is overset. Two items that do require some discussion, however, are the grid file and grid file format used as input to FCONVERT.

When using SUGGAR, it is recommended that the output grid from SUGGAR be used as the input grid to FCONVERT. In this way, any grid modifications that occur during the grid assembly process will be properly reflected in DPLR, and the grid system in DPLR will be consistent with the information contained in the DCI file.

For example, SUGGAR allows for grid blocks to be independently transformed (scaled, translated, and/or rotated) via the SUGGAR input file, a feature that is enabled by and sometimes useful during overset gridding. SUGGAR may also require that blocks be re-ordered within the SUGGAR input file in order to achieve the correct hole-cutting and assembly. In both cases, the composite grid that is output by SUGGAR would differ from the original grid that was input to SUGGAR, and it is the output composite grid that should be input to FCONVERT.

For 3D cases, specifying `inform=2` in the FCONVERT input file will allow FCONVERT to read the unformatted PLOT3D composite grid file output by SUGGAR (`e.g. SUGGAR/allgrids.p3dudl`). For 2D cases, SUGGAR outputs a 3D PLOT3D grid file that specifies `nk=1` in all blocks. A recent modification to FCONVERT (released with DPLR Code Package Version 4.01.1) should allow FCONVERT to read such a file correctly by specifying `idim=2` and `inform=2`, for example. Alternatively, the provided **`convert_to_2d`** utility can be used to convert the SUGGAR composite grid file to a formatted 2D PLOT3D file acceptable to FCONVERT, at which point `inform=22` can be used in FCONVERT to read the resulting formatted file (`e.g. SUGGAR/allgrids.g.`

Generally speaking, run FCONVERT as usual, using `iaction=10` to convert the grid to FXDR format `(ouform=11),` for example, or `iaction=1` if any block splitting is desired (See Chapter 3 for more information on Using FCONVERT).

8.4 Running DPLR

Once the grid and DCI files have been generated, only a few considerations are necessary when running the DPLR flow solver for overset grids.

- For the overset logic to be *available* in DPLR, the DPLR Code Package must be linked with DiRTlib at compile time (see Section 8.1).

- For the overset logic to be *activated*, the iover flag must be set to 1 in the DPLR input file, and the name of the DCI file must be specified. (See the discussion for the Input Filenames and Overset Grid Implementation portions of the DPLR input file in Section 4.2.)

- The boundary condition on overset boundaries should be specified using boundary flag 901.

Tech Tips:
1). *Currently, the only DCI file format that is supported is the flex file generated by SUGGAR. (DiRTlib can automatically detect and read both ASCII and unformatted versions of this file.) As such, the ioint flag in the Overset Grid Implementation section of the DPLR input file is presently ignored.*

2). *The boundary condition on overset boundaries should have no influence on the flow solution. First-order extrapolation is recommended on these boundaries simply to provide reasonable values to the post-processing and visualization codes. Boundary flag 901 has been added beginning with DPLR Code Package Version 4.01.1 to designate this specification for overset boundaries, but within DPLR, boundary flags 3 and 901 are treated identically.*

8.5 Grid Adaption

Extra steps are required when grid adaption is performed on overset grids. Each time that the grid is adapted in DPLR, the domain connectivity information must be recalculated and imported. DPLR currently executes SUGGAR via a system call to a script named `update_suggar`, which must be placed in the main working directory. Example scripts named `update_suggar_2d` and `update_suggar_3d`, to be used with 2D and 3D cases respectively, are provided in the `utilities/overset` directory. Each script performs the following basic steps:

Step 1: **Convert** the adapted output grid from DPLR to a form suitable for SUGGAR using `fconvert`. In 2D cases, this requires the extra step of converting the 2D file to a pseudo-2D file that specifies `nk=1` for all blocks.

Step 2: **Split** the grid into single block files using `p3dconvert`.

Step 3: **Re-run** SUGGAR.

8.6 Post-Processing

Tecplot is the preferred visualization software for overset DPLR solutions due to its ability to handle cell-centered (or primary value) blanking explicitly. Retaining the `iblank` information at the cell centers is recommended because of the fact that the `iblank` values cannot be transferred from the cell centers to the vertices in an entirely meaningful way.

Modifications to the steps normally taken with POSTFLOW during post-processing may be necessary when overlapping grids are involved. The following sections discuss several such recommendations.

8.6.1 Field Plots

Ideally, some variables (e.g. the grid coordinates) should be stored at the vertices while others (e.g. the iblank array and possibly the dependent variables) should be stored at the cell centers. Tecplot accommodates this mixed-mode method of storage, but POSTFLOW currently does not. In the meantime, such a file can be created by the provided `merge_dplr` utility.

Therefore, for 2D or 3D plots of the flow field, run POSTFLOW, choosing `ouform=25` (Tecplot block ASCII format) and `interp=11` (cell centers, no boundaries). Do not include the grid coordinates (`ivarp=0`), and do not include the iblank values (`ivarp=26`) in the variable list. Write all volumes points, and then use the provided `merge_dplr` utility to combine the composite grid (from SUGGAR), iblank values (from the DCI file) and dependent variables (from the POSTFLOW output file). Within Tecplot, activate primary value blanking, blanking cells in which the primary value of iblank is zero (which will disable the OUT cells, possibly leaving some overlap between grids) or less than or equal to zero (which will disable the OUT and FRINGE cells, possibly leaving some gaps between grids).

8.6.2 Surface Integration

If overlapping surface grids are present in the grid system, the panel weights calculated by USURP must be incorporated into any surface integration (e.g. for the calculation of aerodynamic forces or heat transfer).

Step 1: **Run** USURP, providing the DPLR input file by redirection; e.g.

```
usurp < dplr.inp
```

where `dplr.inp` is the name of the DPLR input file of interest. Assuming there were no problems, a `panel_weights.dat` file is created.

Tech Tip: *USURP will obtain the name and format of the grid file, the name of the DCI file (if the* `iover` *flag is set to 1), and the identity of all solid wall boundaries from the DPLR input file.*

Step 2: **Run** POSTFLOW, specifying the normal flags for surface integration, such as `ouform=8`, `interp=11`, and an appropriate value for `ivarp` (e.g. `ivarp=521`; see Chapter 5). If the `panel_weights.dat` file is present, POSTFLOW will automatically read it and apply it to the integrand of any surface integration operation.

Tech Tip: *Hiding the panel_weights.dat file from POSTFLOW (by temporarily changing its name) is one way to verify that the panel weights are being applied to the surface integrations. Comparing values of the wetted surface area calculated by USURP and by POSTFLOW (e.g. by specifying ouform=8, interp=11, iexbc=26 and ivarp=23 in the POSTFLOW input file) is one way to verify that the USURP panel weights are being applied correctly.*

8.6.3 Surface Plots

Contour plots on surfaces comprised of overlapping surface grids can be cleaned up considerably if the overlapping portions are removed. USURP has the ability to remove the overlapping portions and produce a singly-defined surface mesh in its place. The output surface quantities from POSTFLOW can then be transferred to this new surface representation for visualization purposes.

The provided `merge_usurp` utility merges the derived variables from POSTFLOW with the surface mesh definition from USURP to try to generate improved surface plots.

Step 1: **Run** POSTFLOW, specifying `interp=11` (cell-centered values, no boundary points) and `ouform=26` (Tecplot point ASCII format).

Step 2: **Run** USURP, specifying these command line options:

`--tecformat=ascii` (required by `merge_usurp`)

`--watertight` (if possible)

`--basis=patch` (optional)

e.g.,

```
usurp --tecformat=ascii --watertight --basis=patch < dplr.inp
```

(See the documentation included with the USURP distribution for more information.)

Step 3: **Run** `merge_usurp`. If USURP succeeded in generating a watertight triangulation in Step 2 above, then the resulting `usurp-triangles.dat` file should be provided as input to `merge_usurp`. Failing that, the `usurp-surfaces.dat` file can be used instead.

Step 4: **Load** the resulting `merged_surfaces.dat` file into Tecplot to view surface contours of the selected dependent variables.

8.7 Examples

A few simple examples will serve to demonstrate the basic functionality of the overset version of DPLR and the procedures required to set-up, run, and post-process the cases.[2]

8.7.1 Overset 2D Cylinder Case with Tilted Hole Patch

A simple 2D case for testing the overset logic was generated beginning from the `cylinder-ivib1` sample case distributed with DPLR.

Step 1: **Prepare** the grid files. The left-most panel of Figure 8-1 shows the original 101 x 101 mesh for that case (with every other grid point removed in each direction for better clarity). An overset case was manufactured by manipulating the original grid in Gridgen and then using SUGGAR to cut a hole in the original mesh, as shown in the center panel of Figure 8-1. To obtain a continuous solution over the domain, the hole was covered by a patch grid created from a subset of the original grid but rotated 5 degrees, as shown in the right-most panel of Figure 8-1.

[2] All examples were conducted using SUGGAR Version 2.57 and, where applicable, the accompanying version of surfasm, which was version 1.24.

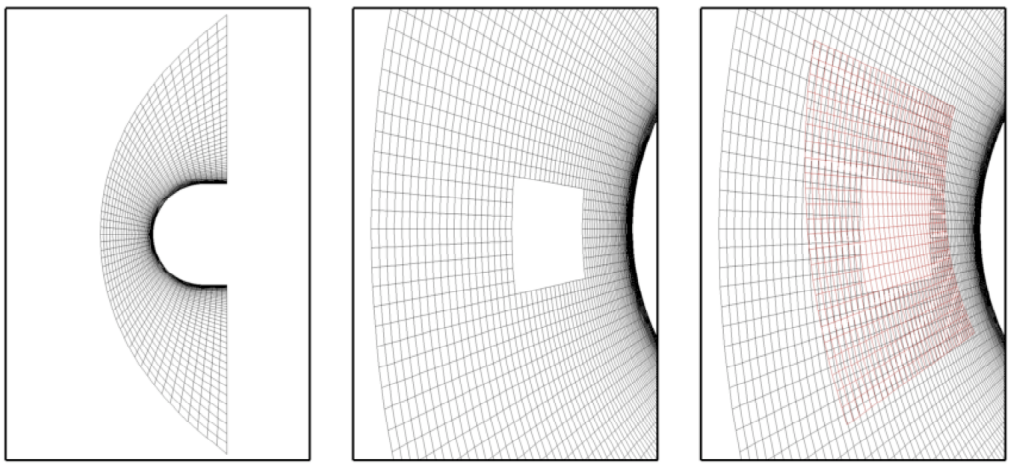


Figure 8-1 Overset Grid for 2D Cylinder Example

Step 2: **Prepare** the SUGGAR input file. The SUGGAR input file was first generated using `gg2suggar`, and then the hole, which is shown in the right-hand panel of Figure 8-1, was created by adding the following line:

```
<blank_region range1="46:54"
     range2="64:80" range3="1:1" mark_centers="yes"/>
```

where `range1` and `range2` refer to the cells (due to the `mark_centers` command) in the i-direction and j-direction respectively. The final SUGGAR input file is shown in Figure 8-2. Because all four boundaries of the patch grid are overset boundaries, no boundary conditions are specified for the patch grid in the SUGGAR input file, as can be seen in the definition for block1 in Figure 8-2.

Step 3: **Run** SUGGAR.

Action: At the command line, type:

```
run_suggar_2d
```

Result: SUGGAR generates the domain connectivity information with no orphans. As specified in the SUGGAR input file,

the domain connectivity information is stored in `gen_dirt.dci`, and the composite grid is stored in `allgrids.p3dudl` as an unformatted, double-precision, PLOT3D file.

Tech Tip: *SUGGAR marks the cells in the hole region with iblank=0 and then marks two layers of fringe cells (iblank < 0) around the hole, where values for the dependent variables must be interpolated. The solution for the governing equations must be obtained in the remaining "active" cells, where iblank=1. The iblank values for the grid assembly are shown in Figure 8-3.*

```
    <global>
    <cell_centered mark_using_neighbors="y"/>
    <fringe_stencil type="diag+planar_first_offdiag"/>
    <output>
       <structured_grid filename="allgrids.p3dudl" style="p3dudl"/>
       <donor_receptor_file filename="gen_dirt.dci"
             style="ascii_gen_drt_pairs"/>
    </output>
    <body name="root body">
<volume_grid name="block1"
          style="p3d" filename="Grids/block_1.grd">
       </volume_grid>
       <volume_grid name="block2"
           style="p3d" filename="Grids/block_2.grd">
          <blank_region range1="46:54" range2="64:80" range3="1:1"
                       mark_centers="yes"/>
          <boundary_surface name="jmin">
             <region range1="all" range2="min" range3="all"/>
             <boundary_condition type="solid"/>
          </boundary_surface>
          <boundary_surface name="imax">
             <region range1="max" range2="all" range3="all"/>
             <boundary_condition type="farfield"/>
          </boundary_surface>
          <boundary_surface name="jmax">
             <region range1="all" range2="max" range3="all"/>
             <boundary_condition type="farfield"/>
          </boundary_surface>
          <boundary_surface name="imin">
             <region range1="min" range2="all" range3="all"/>
             <boundary_condition type="farfield"/>
          </boundary_surface>
       </volume_grid>
    </body>
    </global>
```

Figure 8-2 SUGGAR Input File for 2D Cylinder Example

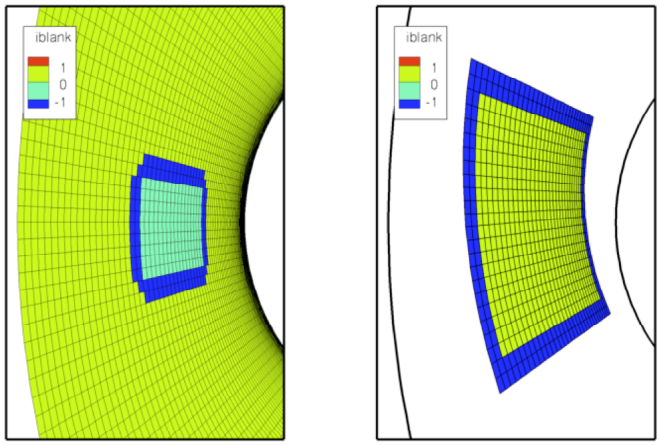


Figure 8-3 Iblank Values for 2D Cylinder Example

Step 4: **Run** FCONVERT (with `iaction=1`, `idim=2`, `inform=2`, `ouform=11`, and `ibrk=jbrk=2` in each block) to convert the `SUGGAR/allgrids.p3dudl` file into an FXDR-formatted grid file named `dplr.pgrx` suitable for parallel execution on 8 processors. (Note that a modification to FCONVERT introduced in DPLR Code Package 4.01.1 is necessary to read this 3D grid with `nk=1` as a 2D grid in FCONVERT.)

Step 5: **Prepare** the DPLR input file. In this case, the DPLR input file was constructed by starting with the `cylinder-ivib1.inp` file that was provided with the "Cylinder" sample case. The input file describes laminar flow of a five-species air model in vibrational non-equilibrium, resulting in nine coupled partial differential equations. The primary modifications to the input file involved changing the number of blocks from 1 to 2 and adding a block-specific section for the patch grid, using `ibc=901` on the overset boundaries. The maximum CFL number was set to 100,000. The overset logic was activated by setting `iover=1` and specifying the name of the DCI file as `SUGGAR/gen_dirt.dci`.

Step 6: **Run** DPLR. In this case, DPLR was executed with the command

```
mpirun -np 8 dplr2d < dplr.inp
```

Over 1000 iterations, the RMS residual dropped 14 orders of magnitude.

Step 7: **Run** POSTFLOW (with `ouform=25`, `interp=11`, and `ivarp=150,151`) to generate ASCII Tecplot data. The POSTFLOW output was combined with the composite grid and DCI files from SUGGAR using `merge_dplr`. Velocity contours are shown for the final solution in Figure 8-4. It can be seen that the contours vary smoothly across the overset region. In the center and right-hand panel of Figure 8-4, the velocity contours are shown again, this time showing each block separately and using a piecewise-constant coloring scheme based on the local value of the velocity in each cell. The values of the dependent variables in the hole cells do not influence the solution in the active cells, so these cells are not shown in the center panel.

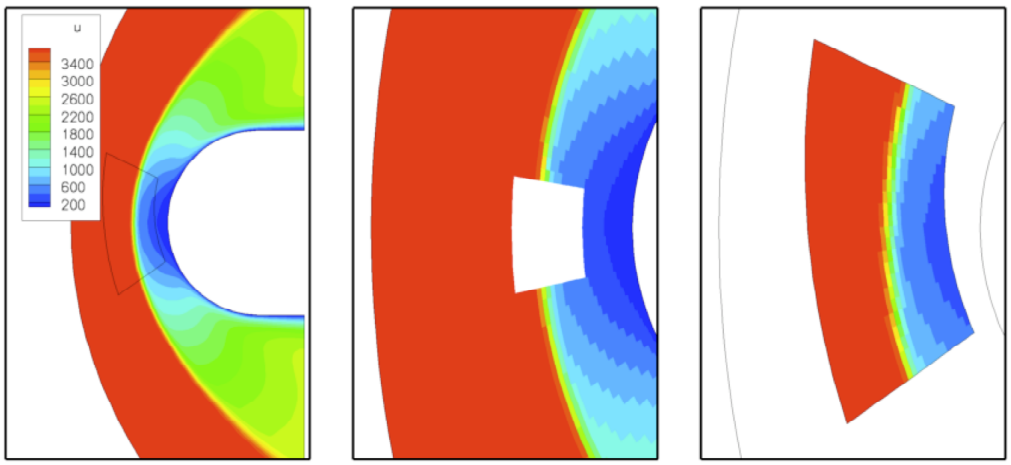


Figure 8-4 Velocity Contours for 2D Cylinder Example

8.7.2 3D MSL Flight Case with Overset Nose Patch

The Mars Science Laboratory Flight sample case that was distributed with DPLR Version 3.05.0 consisted of a two-block point-matched grid, with one rectangular mesh on the nose surrounded by an O-type grid covering the rest of the heat shield, and with both blocks extruded away to the far field. In the example presented here, the central block was removed and replaced by an overset nose patch, which was extruded away to the original far field boundary location, as shown in Figure 8-5.

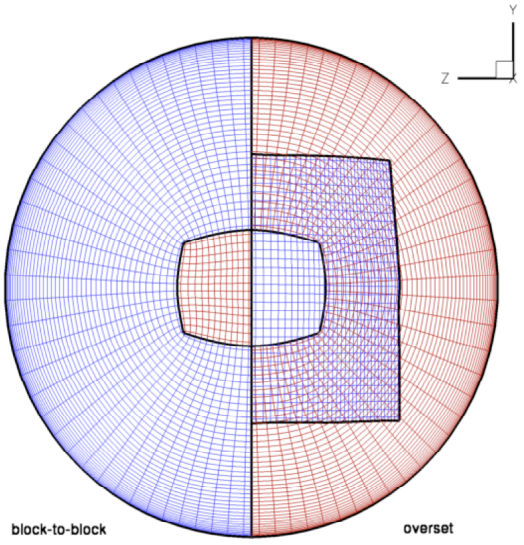


Figure 8-5 Overset Surface Grid for MSL Flight Example

Step 1: **Prepare** the grid files. Working from Gridgen, the 3D volume grid was exported as an unformatted, double-precision, PLOT3D file named `msl-flight-3.grd`, and the generic AS/W boundary conditions were exported to a file named `bc.dat`. The grid file was split into single block, double-precision, unformatted PLOT3D files using the commands

```
mkdir Grids

p3dconvert msl-flight-3.grd -sp3dudl Grids/block.grd
```

which created `block_1.grd` and `block_2.grd` in a `Grids` subdirectory.

Step 2: **Prepare** the SUGGAR input file. In this case, the SUGGAR input file (Input.xml, which is shown in its entirety in Figure 8-6) was generated from the Gridgen output files using `gg2suggar`. The output from `gg2suggar` required only one modification in this case, which was to uncomment the `symmetry` element in the header and specify the symmetry plane as `z`.

```
<global>
<symmetry_plane axis="z"/>
<cell_centered mark_using_neighbors="y"/>
<fringe_stencil type="diag+planar_first_offdiag"/>
<output>
   <structured_grid filename="allgrids.p3dudl" style="p3dudl"/>
   <donor_receptor_file filename="gen_dirt.dci"
        style="ascii_gen_drt_pairs"/>
</output>
<body name="root body">
   <volume_grid name="B"
       style="p3d" filename="Grids/block_1.grd">
      <boundary_surface name="kmin">
         <region range1="all" range2="all" range3="min"/>
         <boundary_condition type="symmetry"/>
      </boundary_surface>
      <boundary_surface name="kmax">
         <region range1="all" range2="all" range3="max"/>
         <boundary_condition type="symmetry"/>
      </boundary_surface>
      <boundary_surface name="imax">
         <region range1="max" range2="all" range3="all"/>
         <boundary_condition type="non-overlap"/>
      </boundary_surface>
      <boundary_surface name="jmin">
         <region range1="all" range2="min" range3="all"/>
         <boundary_condition type="solid"/>
      </boundary_surface>
```

```
      <boundary_surface name="jmax">
         <region range1="all" range2="max" range3="all"/>
         <boundary_condition type="non-overlap"/>
      </boundary_surface>
   </volume_grid>
   <volume_grid name="A"
       style="p3d" filename="Grids/block_2.grd">
      <boundary_surface name="imin">
         <region range1="min" range2="all" range3="all"/>
         <boundary_condition type="symmetry"/>
      </boundary_surface>
      <boundary_surface name="jmin">
         <region range1="all" range2="min" range3="all"/>
         <boundary_condition type="solid"/>
      </boundary_surface>
      <boundary_surface name="jmax">
         <region range1="all" range2="max" range3="all"/>
         <boundary_condition type="non-overlap"/>
      </boundary_surface>
   </volume_grid>
</body>
</global>
```

Figure 8-6 SUGGAR Input File for MSL Flight Example

Step 3: **Run** SUGGAR.

Action: At the command line, type:

```
run_suggar_3d
```

Result: SUGGAR generates the domain connectivity information with no orphans. As specified in the SUGGAR input file, the domain connectivity information is stored in

`gen_dirt.dci`, and the composite grid is stored in `allgrids.p3dudl` as an unformatted, double-precision, PLOT3D file.

Step 4: **Run** FCONVERT (with iaction=1, idim=3, inform=2, ouform=11) to convert `SUGGAR/allgrids.p3dudl` to an FXDR-formatted grid file named `split.pgrx` suitable for parallel execution on 11 processors. For the decomposition, `(ibrk,jbrk,kbrk)=(7,1,1)` for block 1 and `(4,1,1)` for block 2.

Step 5: **Prepare** the DPLR input file. In this case, the DPLR input file was modified from the original sample case with the new block-specific information for the overset nose patch and using `ibc=901` on the overset boundaries. The laminar flow model consisted of an 8-species model of the atmosphere on Mars (beginning with 97.088% CO_2 and 2.912% N_2) in vibrational non-equilibrium, leading to 13 partial differential equations. The maximum CFL number was set to 100,000. The overset logic was activated by setting `iover=1` and specifying the name of the DCI file as `SUGGAR/gen_dirt.dci`.

Step 6: **Run** DPLR. In this case, DPLR was executed with the command

```
mpirun -np 11 dplr3d < dplr.inp
```

Over 800 iterations, the RMS residual dropped 10 orders of magnitude.

Step 7: **Run** USURP.

Action: At the command line, type:

```
usurp --basis=patch --never-skip < dplr.inp
```

Result: The data file `panel_weights.dat` and the Tecplot file `usurp-surfaces.plt` are created.

The left-hand panel of Figure 8-7 shows the resulting mesh from `usurp-surfaces.plt`, in which a portion of the original heat shield mesh has been removed in favor of the finer nose patch mesh. Cells that are partially revealed at the interface have been replaced by black triangles for visualization purposes. The right-hand panel of Figure 8-7 shows the value of the panel weight in each cell (omitting cells with panel weights less than 0.001). The panel weight is a scale factor that multiplies the contribution on each cell face to any subsequent surface integration. The panel weights in the nose patch are all 1.0 in this case, so cells in the heat shield

mesh have panel weights of zero in the overlap region or panel weights less than 1 in the interface region.

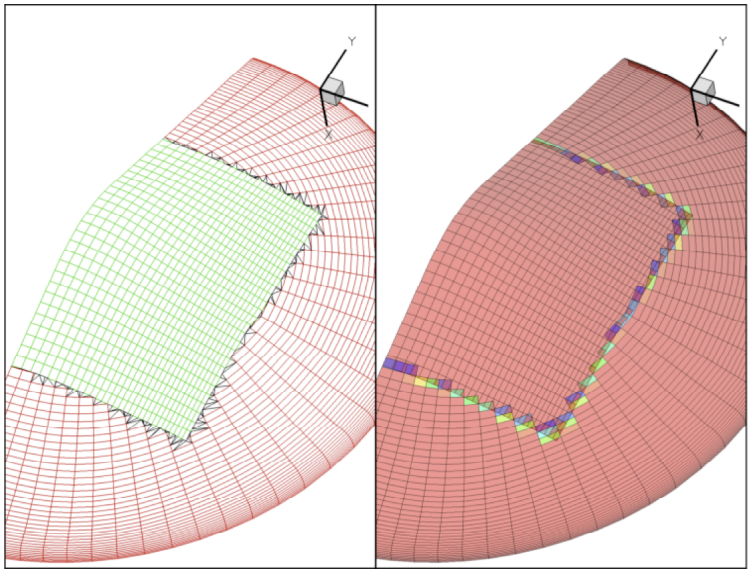


Figure 8-7 Surface Integration Grid from USURP

In this case, USURP reported that the wetted area of the two surface grids was 2.837, representing a 23% reduction from the value obtained if the area integration was conducted without consideration for the overlapping region.

Step 8: **Run** POSTFLOW (with `ouform=26`, `interp=11`, and `ivarp=150,151`) to generate surface contour plots, following the procedure outlined in Section 8.6.3. A comparison of the pressure and temperature for the block-to-block and overset cases is presented in Figure 8-8. A comparison of the heat flux and wall shear stress for the block-to-block and overset grids is shown in Figure 8-9.

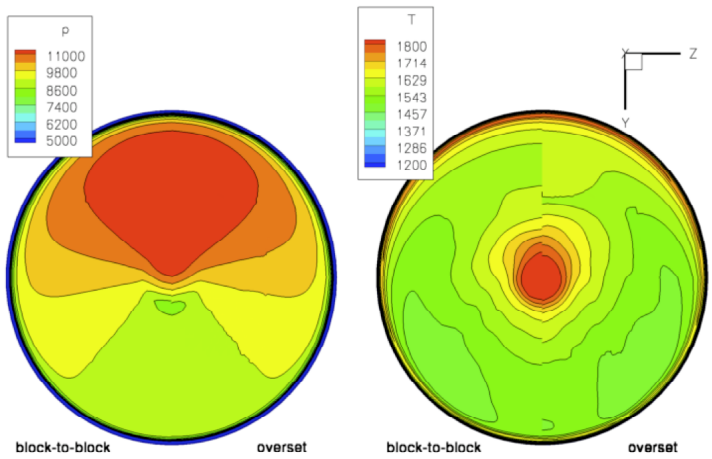


Figure 8-8 Comparison of Solutions for Surface Pressure and Temperature

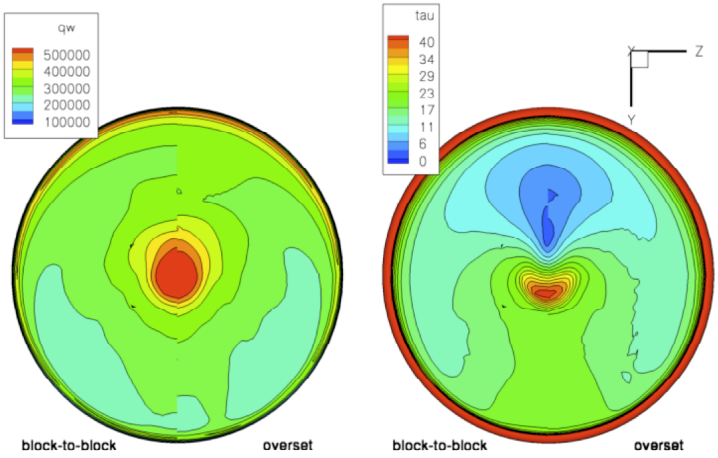


Figure 8-9 Comparison of Solutions for Surface Heat Flux and Shear Stress

Step 9: **Run** POSTFLOW (with `ouform=8` and `interp=11`) to generate integrated surface data. For this case, `iexbc=26` and `ivarp=23,531,611,621` to extract the total wetted area, total heating, pressure force in the x-direction, and viscous force in the x-direction, respectively. The results with and without the panel_weights.dat file present are presented in Table 8.1, where they are compared to those from the original block-to-block sample case.

Table 8.1 – Effect of Panel Weights on Surface Integrals

Quantity	Overset, without panel weights	Overset, with panel weights	Block-to-Block
Area (m^2)	3.693	2.839	2.837
Heat Flux (W)	1.063E+06	8.181E+05	8.160E+05
x-component of pressure force (N)	3.221E+04	2.402E+04	2.401E+04
x-component of viscous force (N)	34.39	31.66	31.75

8.7.3 Huygens-PH Example with 2D Grid Alignment

The Huygens probe, built by the European Space Agency, touched down on Titan, Saturn's largest moon, on January 14, 2005 after a journey of more than seven years. The simulation here, which stems from the Huygens-PH example distributed with DPLR Version 3.05.0, models a point of the descent at which the probe is traveling at 5.126 km/s. The atmosphere of Titan was modeled with 26 chemical reactions among

14 species, beginning from 97.65% nitrogen, 1.02% argon, and 1.33% methane at 176.6 K.

Step 1: **Prepare** the grid files. For demonstration purposes, a two-dimensional, single-block grid was created in Gridgen using hyperbolic extrusion from a 61-point connector defining the heat shield, using an initial spacing of 32 microns, 64 steps, and a stretching ratio of 1.15. This initial 61 x 65 grid was then split into two overlapping 35 x 65 blocks, providing 8 cells of perfect overlap. The two-block 2D grid was exported as a volume grid to an unformatted, double-precision, PLOT3D file named `huygens-overset.grd`. The grid file was split into two single-block, double-precision, unformatted PLOT3D files for SUGGAR using the commands

```
mkdir Grids

p3dconvert huygens-overset.grd-sp3dudl Grids/block.grd
```

which created `block_1.grd` and `block_2.grd` in the `Grids` subdirectory.

Step 2: **Prepare** the SUGGAR input file. The file `Input.xml` for this case was created automatically using `gg2suggar` and is shown in its entirety in Figure 8-10.

```
<global>
<cell_centered mark_using_neighbors="y"/>
<fringe_stencil type="diag+planar_first_offdiag"/>
<output>
   <structured_grid filename="allgrids.p3dudl" style="p3dudl"/>
   <donor_receptor_file filename="gen_dirt.dci"
        style="ascii_gen_drt_pairs"/>
</output>
<body name="root body">
    <volume_grid name="A"
       style="p3d" filename="Grids/block_1.grd">
    <boundary_surface name="jmin">

         <region range1="all" range2="min" range3="all"/>
```

```
            <boundary_condition type="solid"/>
         </boundary_surface>
         <boundary_surface name="jmax">
            <region range1="all" range2="max" range3="all"/>
            <boundary_condition type="farfield"/>
         </boundary_surface>
         <boundary_surface name="imin">
            <region range1="min" range2="all" range3="all"/>
            <boundary_condition type="axis"/>
         </boundary_surface>
      </volume_grid>
      <volume_grid name="B"
          style="p3d" filename="Grids/block_2.grd">
         <boundary_surface name="jmin">
            <region range1="all" range2="min" range3="all"/>
            <boundary_condition type="solid"/>
         </boundary_surface>
         <boundary_surface name="imax">
            <region range1="max" range2="all" range3="all"/>
            <boundary_condition type="farfield"/>
         </boundary_surface>
         <boundary_surface name="jmax">
            <region range1="all" range2="max" range3="all"/>
            <boundary_condition type="farfield"/>
         </boundary_surface>
      </volume_grid>
   </body>
   </global>
```

Figure 8-10 SUGGAR Input File for Huygens Example Case

Step 3: **Run** SUGGAR.

Action: At the command line, type:

```
run_suggar_2d
```

Result: SUGGAR generates the domain connectivity information with no orphans. As specified in the SUGGAR input file, the domain connectivity information is stored in `gen_dirt.dci`, and the composite grid is stored in `allgrids.p3dudl` as an unformatted, double-precision, PLOT3D file.

Step 4: **Run** FCONVERT (with `iaction=10`, `idim=2`, `inform=2`, `ouform=11`) to convert the `SUGGAR/allgrids.p3dudl` file into an FXDR-formatted grid file named `dplr.pgrx`. (Note that a modification to FCONVERT introduced in Version 4.01.1 is necessary to read this 3D grid with `nk=1` as a 2D grid in FCONVERT.)

Step 5: **Prepare** the DPLR input file. In this case, the input file was constructed by starting with the `Nov11HR-t189s.inp` input file that was distributed with the DPLR Version 3.05.0 sample cases and updating it to DPLR Version 4.01.0 using the `dpconvert` utility in `$DPLR_HOME/utilities` by executing the command

```
dpconvert -i Nov11HR-t189s.inp -o dplr.inp
```

The simulation used the `titan14sp-gokcen.chem file`, a 14-species model of Titan's atmosphere as discussed earlier, and assumed vibrational non-equilibrium using a two-temperature model (`ivib=4`), resulting in a system of 18 partial differential equations for this 2D case. The maximum CFL number was set to 3000. For the overset grid case, a block-specific section was added for the second block, and the boundary condition flag on the two overset boundaries was set to 901. The overset logic was activated by setting `iover=1` and specifying the name of the DCI file as `SUGGAR/gen_dirt.dci`.

Step 6: **Run** DPLR. In this case, DPLR was executed with the command

```
mpirun -np 2 dplr2d < dplr.inp
```

Over 1200 iterations, the RMS residual dropped 9 orders of magnitude.

Step 7: **Prepare** the DPLR and SUGGAR input files for grid adaption. The DPLR input file was next modified to enable two rounds of grid alignment with 500 iterations in between. The grid alignment flags are shown in Figure 8-11 below. The provided `update_suggar_2d` script was copied to a file named `update_suggar` and placed in the working directory.

Each time that DPLR adapts the grid, `update_suggar` is executed to generate a new DCI file which is then imported by DPLR.

```
==================================
Grid Adaption
==================================

      igalign      ngiter      nalign      i1stadpt
      1              500          2           1

      imedge      imradial      ngeom      ismooth
      1               1           2           3

      fs_scale      ds_mult      gmargin
      0.92            2.5          0.0

      ds1      cellRe      ds1mx      ds2fr
      0.0       1.0      1.0d-4       0.3

==================================
```

Figure 8-11 Grid Adaption Flags

Step 8: **Run** POSTFLOW (with `ouform=25`, `interp=11`, and `ivarp=110`) to generate ASCII Tecplot data. The POSTFLOW output was combined with the composite grid and DCI files from SUGGAR using `merge_dplr`. Pressure contours are shown in Figure 8-12 for the two-block overset grid before and after alignment. The figure includes the (red and green) boundaries of the two overlapping grid blocks.

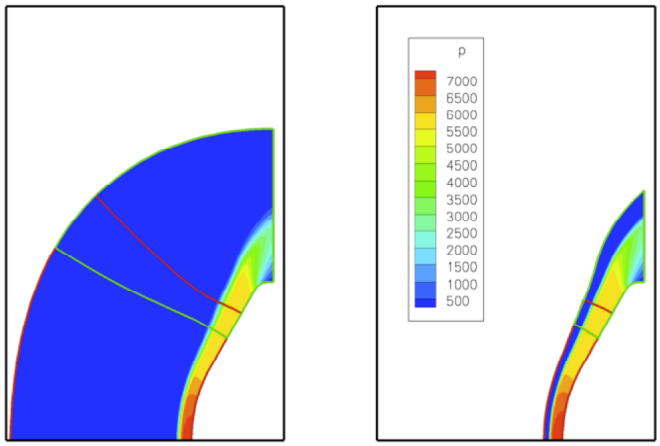


Figure 8-12 Pressure Contours for Huygens Grid Adaption Example

8.7.4 3D MSL Flight Example with Grid Alignment

In Section 8.7.2, the MSL Flight example that was distributed with DPLR Version 3.05.0 used a grid that had already been aligned with the expected location of the shock. The overset nose patch that was added was built to match this pre-determined free stream grid spacing and location. This section presents the results when the overset grid is rebuilt without prior knowledge of the shock location and then subsequently aligned to the solution.

Step 1: **Prepare** the grid files. Using Gridgen, the surface mesh from Section 8.7.2 was extruded hyperbolically using 64 steps with an initial spacing of 5.4 microns and a stretching ratio of 1.15. The resulting volume grid was exported to an unformatted, double-precision, PLOT3D file named `msl-adapt.grd`, and the boundary conditions were exported in the generic AS/W format to a file named `bc.dat`. The grid file was split into two single-block, double-precision, unformatted PLOT3D files for SUGGAR using the commands.

```
mkdir Grids

p3dconvert msl-flight-adapt.grd -sp3dudl Grids/block.grd
```

which created `block_1.grd` and `block_2.grd` in the `Grids` subdirectory.

Step 2: **Prepare** the SUGGAR input file. The SUGGAR input file was generated from the Gridgen output files using `gg2suggar`. As in Section 8.7.2, the file `Input.xml` produced by `gg2suggar` required only one modification, which was to uncomment the `symmetry` element in the header and specify the symmetry plane as `"z"`.

Step 3: **Run** SUGGAR.

Action: At the command line, type:

```
run_suggar_3d
```

Result: SUGGAR generates the domain connectivity information with 59 orphans. As specified in the SUGGAR input file, the domain connectivity information is stored in `gen_dirt.dci`, and the composite grid is stored in `allgrids.p3dudl` as an unformatted, double-precision, PLOT3D file.

The left-hand panel of Figure 8-13 shows the cell-centered iblank values for this grid prior to adaption, where the fringe cells are shaded blue and the orphan cells are shaded red. In this case, the orphans result from the fact that the hyperbolic extrusion did not extend the nose cap grid, which has a cyan border, as far as the main heat shield grid, which has a purple border, such that not all fringe cells in the heat shield grid had valid donors. As the two grid blocks were later aligned with the shock, this offset and the associated number of orphans decreased, from 40 orphans after the first alignment to 19 after the second, which is shown in the right-hand panel of Figure 8-13.

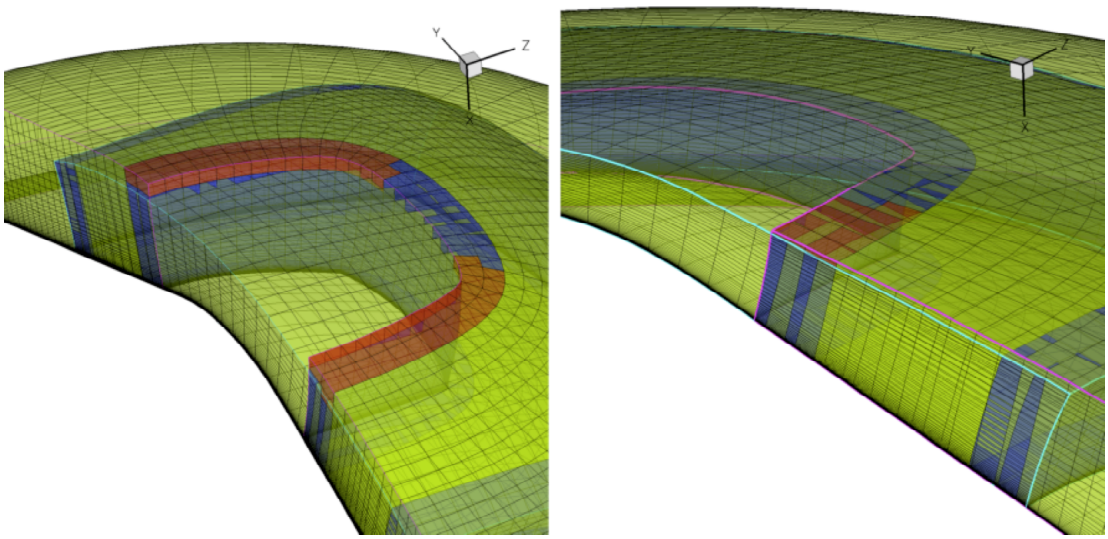


Figure 8-13 Iblank Values Before and After Adaption

Step 4: **Run** FCONVERT (with `iaction=1`, `idim=3`, `inform=2`, `ouform=11`) to convert `SUGGAR/allgrids.p3dudl` to an FXDR-formatted grid file named `dplr.pgrx` suitable for parallel execution on 11 processors. For the decomposition, `(ibrk,jbrk,kbrk)=(7,1,1)` for block 1 and `(4,1,1)` for block 2.

Step 5: **Prepare** the DPLR input file. The input file was prepared in the same manner as in Section 8.7.2. The boundary flag on overset boundaries was again set to 901, and the overset logic was activated by setting `iover=1` and specifying the name of the DCI file as `SUGGAR/gen_dirt.dci`.

Step 6: **Run** DPLR. In this case, DPLR was executed with the command

```
mpirun -np 11 dplr3d < dplr.inp
```

An initial solution was obtained on the original grid using 800 iterations and a maximum CFL number of 3,000. Subsequently, two grid adaption steps were performed using the same settings as those shown in Figure 8-11, again with a maximum CFL number of 3,000.

Step 7: **Run** POSTFLOW. Surface integration for wetted area, total heating, pressure force, and viscous force was repeated using USURP and

POSTFLOW as described in Section 8.7.2. Those values for the current simulation are shown in Table 8.2 below and again agree with the original block-to-block solution to within 1%.

Table 8.2 Comparison of Overset and Block-to-Block Surface Integrals

Quantity	Overset, with panel weights	Block-to-Block	Difference (%)
Area (m^2)	2.837	2.837	-
Heat Flux (W)	8.191E+05	8.160E+05	0.37
x-component of pressure force (N)	2.394E+04	2.401E+04	-0.29
x-component of viscous force (N)	31.83	31.75	0.25

8.7.5 2D ARD Capsule Example

The Atmospheric Re-entry Demonstrator (ARD) was an unmanned Apollo-like capsule launched by ESA in October 1998. After performing a sub-orbital flight (830 km apogee), the capsule splashed down in the Pacific Ocean and was recovered by the French Navy. Thus, the ARD was the first European spacecraft ever to be successfully recovered. The ARD was 2.8 meters in diameter and was characterized by a spherical blunt nose (R = 3.36 meters), a conical back shell with a 33° half angle, and a back cap that housed the flotation balloons [5].

Step 1: **Prepare** the grid files. An overset grid for a 2D (axisymmetric) representation of the ARD was constructed using hyperbolic extrusion from the spherical nose and conical back shell, as shown in Figure 8-14 (top left). Subsequently, a grid was added to represent the wetted area of the back cap, shown as the blue mesh in the top right portion of Figure 8-14. Finally, a third block, shown as the green mesh in the bottom

left portion of Figure 8-14, was added as a collar grid to cover the junction between the back shell and cap. As described next, SUGGAR was used to cut away the portion of red mesh interior to the back cap, leaving the final assembly as shown in the bottom right portion of Figure 8-14. The grid has a near wall spacing of 32 microns, a stretching ratio of 1.2, and 65 points in the body normal direction, leading to a far-field boundary that was approximately 16 meters from the capsule in all directions.

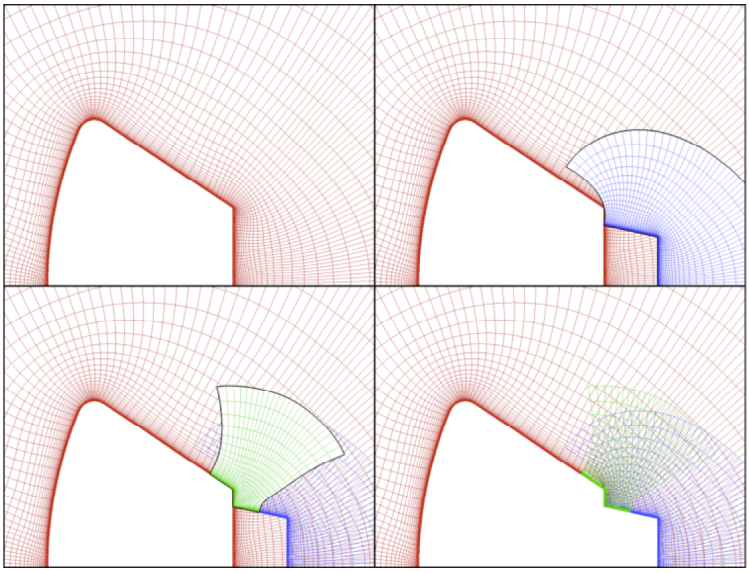


Figure 8-14 Overset Grid System for ARD Example

Step 2: **Prepare** the SUGGAR input file. As a starting point, a SUGGAR input file was generated using `gg2suggar`. The final input file, which is shown in Figure 8-15, resulted after several modifications. First, the `symmetry` element at the top of the file was activated (uncommented) and the symmetry axis was specified as `"y"`. This change causes the geometry to appear to SUGGAR to be watertight, which is necessary for the hole-cutting step to be successful. Second, the root body was divided into two child bodies, with the first block comprising the cone body and the two smaller blocks comprising the frustrum body. Finally, the `jmin` boundary of the collar grid was divided into two portions, and one of the portions was marked as `collar` to the cone body.

```
<global>
<symmetry_plane axis="y"/>
<cell_centered mark_using_neighbors="y"/>
<fringe_stencil type="diag+planar_first_offdiag"/>
<output>
   <structured_grid filename="allgrids.p3dudl" style="p3dudl"/>
   <donor_receptor_file filename="gen_dirt.dci"
       style="ascii_gen_drt_pairs"/>
</output>
<body name="root body">
   <body name="cone">
      <volume_grid name="A"
          style="p3d" filename="Grids/block_1.grd">
      <boundary_surface name="jmin">
         <region range1="1:65" range2="min" range3="all"/>
         <boundary_condition type="solid"/>
      </boundary_surface>
      <boundary_surface name="jmax">
         <region range1="1:65" range2="max" range3="all"/>
         <boundary_condition type="farfield"/>
      </boundary_surface>
      <boundary_surface name="imin">
         <region range1="min" range2="all" range3="all"/>
         <boundary_condition type="symmetry"/>
      </boundary_surface>
      <boundary_surface name="jmin:2">
         <region range1="65:-1" range2="min" range3="all"/>
         <boundary_condition type="solid"/>
      </boundary_surface>
      <boundary_surface name="imax">
         <region range1="max" range2="all" range3="all"/>
         <boundary_condition type="symmetry"/>
      </boundary_surface>
      <boundary_surface name="jmax:2">
         <region range1="65:-1" range2="max" range3="all"/>
         <boundary_condition type="farfield"/>
      </boundary_surface>
   </volume_grid>
</body>
<body name="frustrum">
   <volume_grid name="B"
```

```
        style="p3d" filename="Grids/block_2.grd">
      <boundary_surface name="jmin">
         <region range1="1:17" range2="min" range3="all"/>
         <boundary_condition type="solid"/>
         <collar body="cone"/>
      </boundary_surface>
      <boundary_surface name="jmin:2">
         <region range1="17:-1" range2="min" range3="all"/>
         <boundary_condition type="solid"/>
      </boundary_surface>
</volume_grid>
<volume_grid name="C"
       style="p3d" filename="Grids/block_3.grd">
<boundary_surface name="jmin">
         <region range1="1:25" range2="min" range3="all"/>
         <boundary_condition type="solid"/>
      </boundary_surface>
<boundary_surface name="jmin:2">
         <region range1="25:-1" range2="min" range3="all"/>
         <boundary_condition type="solid"/>
      </boundary_surface>
<boundary_surface name="imax">
         <region range1="max" range2="all" range3="all"/>
         <boundary_condition type="symmetry"/>
      </boundary_surface>
</volume_grid>
 </body>
</body>
</global>
```

Figure 8-15 SUGGAR Input File for ARD Example

Step 3: **Run** SUGGAR.

Action: At the command line, type:

```
run_suggar_2d
```

Result: SUGGAR generates the domain connectivity information with no orphans. As specified in the SUGGAR input file, the domain connectivity information is stored in

`gen_dirt.dci`, and the composite grid is stored in `allgrids.p3dudl` as an unformatted, double-precision, PLOT3D file.

Figure 8-16 shows the IBLANK values of the cells in each of the three blocks and the full assembly, where the dark blue cells are fringe cells and the cyan cells within the back cap are hole cells, which were eliminated by SUGGAR.

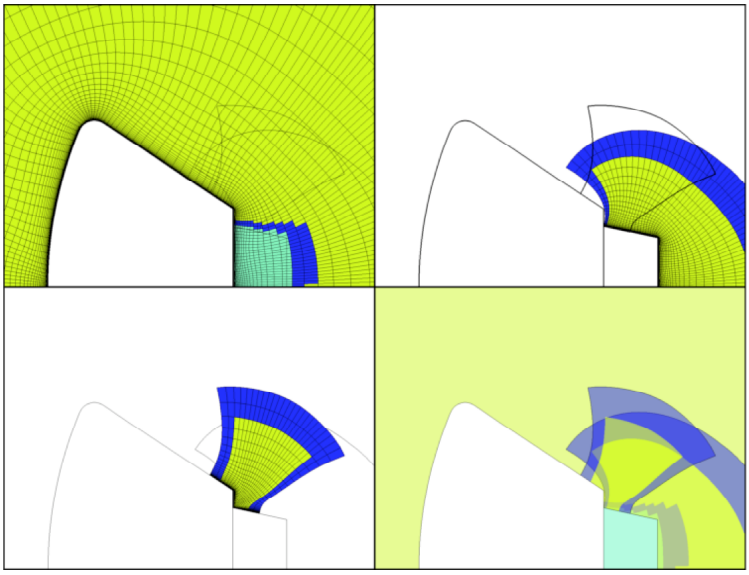


Figure 8-16 Iblank Values for ARD Example

Step 4: **Run** FCONVERT (with `iaction=10`, `idim=2`, `inform=2`, `ouform=11`) to convert the `SUGGAR/allgrids.p3dudl` file into an FXDR-formatted grid file named `dplr.pgrx`. (Note that a modification to FCONVERT introduced in Version 4.01.1 is necessary to read this 3D grid with `nk=1` as a 2D grid in FCONVERT.)

Step 5: **Prepare** the DPLR input file. The DPLR input file specified molecular nitrogen as a perfect gas with $M_\infty = 2.0$, $T_\infty = 219$ K, and a density corresponding to $p_\infty = 2891$ Pa. The maximum CFL number was set to 100,000.

Step 6: **Run** DPLR. In this case, DPLR was executed with the command

```
mpirun -np 3 dplr2d < dplr.inp
```

Over 1000 iterations, the RMS residual dropped 5 orders of magnitude, which was most likely limited by unsteadiness in the large separated region aft of the capsule.

Step 7: **Run** POSTFLOW (with `ouform=25`, `interp=11`, `ivarp=150,151,154`) and `merge_dplr`, following the procedure outlined in Section 8.6.1. The streamlines and Mach number contours from the overset solution are shown in Figure 8-17.

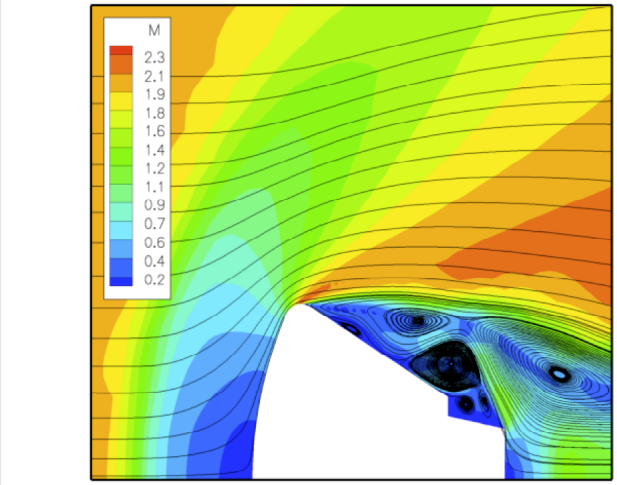


Figure 8-17 Mach Number Contours and Streamlines for ARD Example

8.7.6 2D DART Capsule Example

The Delft Aerospace Re-entry Test (DART) demonstrator was designed as a test-bed for re-entry measurements. It is an axisymmetric, spherical blunt-cone/flare configuration with an overall length of 1.63 meters, maximum diameter of 2.03 meters, and nose radius of 0.51 meters.

Two features set this example apart from the previous one. First, the overset grid is comprised of two blocks, with one block hyperbolically extruded from the body itself and a second Cartesian background block that extends to the far-field and downstream along the wake. Second, the overlap minimization is activated in order to improve the fringe locations.

Step 1: **Prepare** the grid files. The grid was created in Gridgen. The first block was created by hyperbolic extrusion using a near-wall spacing of 32 microns, a stretching ratio of 1.2, and 53 steps, with symmetry boundary conditions applied along the *x*-axis. The second block was created using a uniformly spaced, rectangular grid, which extended from 3.6 meters upstream of the nose to 11 meters downstream of the back of the capsule, and out to a radius of just over 4 meters.

Step 2: **Prepare** the SUGGAR input file. The SUGGAR input file was created using `gg2suggar`. Only three modifications were needed. A body needed to be created to contain each block, in order that the blocks would cut one another, the `symmetry` element needed to be activated and the axis set to `y`, and the `minimize_overlap` element needed to be activated. The resulting file is shown in Figure 8-18.

```
<global>

<symmetry_plane axis="y"/>
<cell_centered mark_using_neighbors="y"/>
<fringe_stencil type="diag+planar_first_offdiag"/>
<minimize_overlap set_dsf="peg5"
     rm_fringe_from_donors="yes"/>
<output>
   <structured_grid filename="allgrids.p3dudl" style="p3dudl"/>
   <donor_receptor_file filename="gen_dirt.dci"
        style="ascii_gen_drt_pairs"/>
</output>
<body name="root body">
   <body name="capsule">
      <volume_grid name="A"
          style="p3d" filename="Grids/block_1.grd">
         <boundary_surface name="jmin">
             <region range1="1:25" range2="min" range3="all"/>
             <boundary_condition type="solid"/>
         </boundary_surface>
         <boundary_surface name="imin">
            <region range1="min" range2="all" range3="all"/>
            <boundary_condition type="symmetry"/>
         </boundary_surface>
         <boundary_surface name="jmin:2">
            <region range1="25:49" range2="min" range3="all"/>
            <boundary_condition type="solid"/>
```

```
            </boundary_surface>
            <boundary_surface name="jmin:3">
               <region range1="49:73" range2="min" range3="all"/>
               <boundary_condition type="solid"/>
            </boundary_surface>
            <boundary_surface name="jmin:4">
               <region range1="73:-1" range2="min" range3="all"/>
               <boundary_condition type="solid"/>
            </boundary_surface>
            <boundary_surface name="imax">
               <region range1="max" range2="all" range3="all"/>
               <boundary_condition type="symmetry"/>
            </boundary_surface>
      </volume_grid>
   </body>
   <body name="background">
      <volume_grid name="B"
          style="p3d" filename="Grids/block_2.grd">
         <boundary_surface name="imin">
            <region range1="min" range2="all" range3="all"/>
            <boundary_condition type="farfield"/>
         </boundary_surface>
         <boundary_surface name="jmax">
            <region range1="all" range2="max" range3="all"/>
            <boundary_condition type="farfield"/>
         </boundary_surface>
         <boundary_surface name="imax">
            <region range1="max" range2="all" range3="all"/>
            <boundary_condition type="farfield"/>
         </boundary_surface>
         <boundary_surface name="jmin">
            <region range1="all" range2="min" range3="all"/>
            <boundary_condition type="symmetry"/>
         </boundary_surface>
      </volume_grid>
    </body>
   </body>
   </global>
```

Figure 8-18 SUGGAR Input File for DART Example

Step 3: **Run** SUGGAR.

Action: At the command line, type:

```
run_suggar_2d
```

Result: SUGGAR generates the domain connectivity information with no orphans. As specified in the SUGGAR input file, the domain connectivity information is stored in `gen_dirt.dci`, and the composite grid is stored in `allgrids.p3dudl` as an unformatted, double-precision, PLOT3D file.

The resulting composite grid is shown in the left-hand panel of Figure 8-19. A hole has been cut in the background (green) block, and the ragged boundaries created by the overlap minimization can be seen. The right-hand panel of Figure 8-19 shows the cell-centered iblank values for this grid. In this figure, the hole cells are disabled, the fringe cells are the darker, blue cells, and the field cells are the yellow cells. It can be seen that the minimization has caused the outer boundary fringe cells for the body grid to be moved inward toward the body, away from the black circular boundary that would otherwise have denoted the outer boundary of that block.

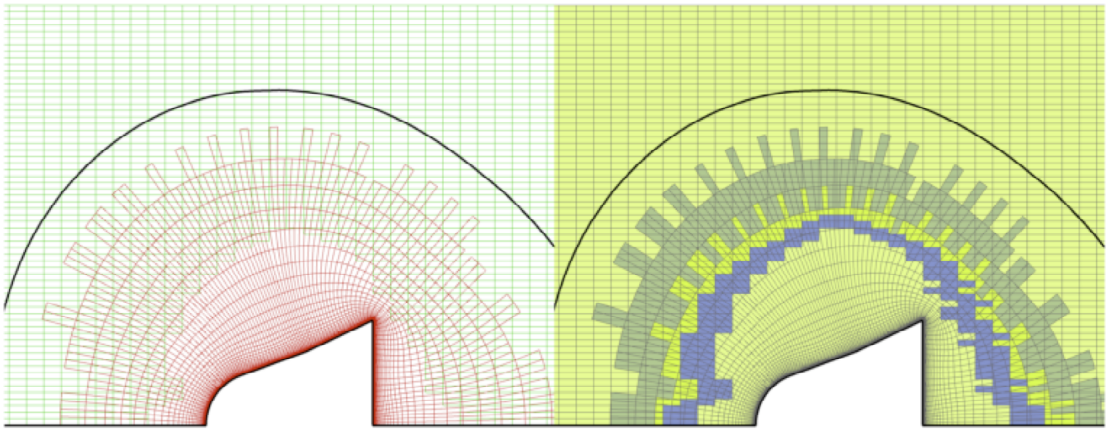


Figure 8-19 Overset Grid and Iblank Values for DART Example

Step 4: **Run** FCONVERT (with `iaction=1`, `idim=2`, `inform=2`, `ouform=11`) to convert the `SUGGAR/allgrids.p3dudl` file into an FXDR-formatted grid file named `dplr.pgrx` suitable for parallel execution on 14 processors. (Note that a modification to FCONVERT introduced in Version 4.01.1 is necessary to read this 3D grid with `nk=1` as a 2D grid in FCONVERT.) For the decomposition, `(ibrk,jbrk)` was set to `(2,1)` for block 1 and `(3,4)` for block 2.

Step 5: **Prepare** the DPLR input file. As with the ARD capsule example, the DART capsule is simulated using molecular nitrogen as a perfect gas at $M_\infty = 2.0$, $T_\infty = 219$ K, and a density corresponding to $p_\infty = 2891$ Pa. The maximum CFL number was set to 100,000. The boundary condition flag on the overset boundary was set to 901. The overset logic was activated by setting `iover=1` and specifying the name of the DCI file as `SUGGAR/gen_dirt.dci`.

Step 6: **Run** DPLR. In this case, DPLR was executed with the command

```
mpirun -np 14 dplr2d < dplr.inp
```

Over 1000 iterations, the RMS residual dropped just over 6 orders of magnitude.

Step 7: **Run** POSTFLOW (with `ouform=25`, `interp=11`, `ivarp=154`) and `merge_dplr`, following the procedure outlined for Field Plots in Section 8.6. The resulting Mach number contours and shown in Figure 8-20.

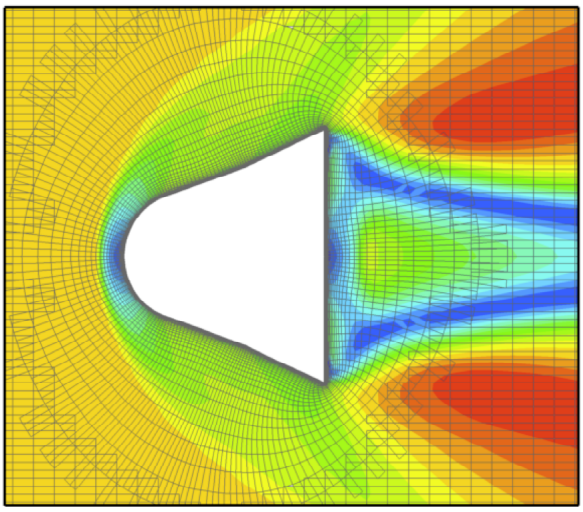


Figure 8-20 Mach Number Contours for DART Example

8.7.7 3D Capsule Example

The final example considers the case of a capsule mounted on a sting, as it might be for a wind tunnel test. The sting holds the capsule such that the heat shield is tilted 28 degrees relative to the flow.

Step 1: **Prepare** the grid files. An axisymmetric grid was first built for the capsule, using a connector containing 97 points along the capsule contour, excluding the axis of rotation to avoid any singularities. A domain was hyperbolically extruded using an initial spacing of 32 microns, a growth rate of 1.2, and 64 steps, such that the domain extended more than 16 meters from the capsule. That domain was then rotated using 33 points through 180 degrees to create the volume. Overset patches were manually constructed to cover the centerline region at the top and bottom of the capsule.

A domain was constructed on the sting surface (with 49 points along the sting and 33 around the half-circumference) that was conformal to the outer boundary created by the capsule grid at one end and that protruded into the capsule at the other end. A volume grid for the sting was then created by normal extrusion in Gridgen, using a

near-wall spacing of 30 microns and a stretching ratio of 1.23, while constraining the outer boundary to remain projected to the far-field boundary from the capsule body grid. This volume grid extended more than 8 meters from the sting surface, which was helpful in covering the hole that the sting cut into the relatively coarse far-field region of the capsule body grid. A collar grid was added to cover the region where the sting and capsule meet.

Step 2: **Prepare** the SUGGAR input file. For the grid cutting to occur, the capsule body grids and the sting grid were divided into two separate bodies in the SUGGAR input file, which is shown in Figure 8-21. In addition, cutting surfaces (`structured_cutting_surface` elements at the end of Figure 8-21) were provided to SUGGAR to close both ends of the sting in order to create a closed cutting surface. Initial runs with SUGGAR subsequently revealed the need for two interface grids to increase the amount of overlap in this region, probably due to the difficulty of growing a large collar grid on the concave side of the sting/capsule intersection.

```
<global>
<donor_quality value="0.8"/>
<symmetry_plane axis="z"/>
<cell_centered mark_using_neighbors="n"/>
<minimize_overlap set_dsf="peg5"  rm_fringe_from_donors="yes" />
<output>
   <structured_grid filename="allgrids.p3dudl" style="p3dudl"/>
   <donor_receptor_file filename="gen_dirt.dci"
         style="ascii_gen_drt_pairs"/>
</output>
<body name="root body">
   <body name="capsule">
      <volume_grid name="capsule"
          style="p3d" filename="Grids/block_1.grd">
         <boundary_surface name="kmin" const_coord="z=0">
            <region range1="1:-1" range2="1:-1" range3="1:1"/>
            <boundary_condition type="symmetry"/>
         </boundary_surface>
         <boundary_surface name="kmax" const_coord="z=0">
            <region range1="1:-1" range2="1:-1" range3="-1:-1"/>
            <boundary_condition type="symmetry"/>
         </boundary_surface>
```

```
   <boundary_surface name="jmin">
      <region range1="1:-1" range2="1:1" range3="1:-1"/>
      <boundary_condition type="solid"/>
   </boundary_surface>
   <boundary_surface name="jmax">
      <region range1="1:-1" range2="-1:-1" range3="1:-1"/>
      <boundary_condition type="farfield"/>
   </boundary_surface>
</volume_grid>
<volume_grid name="bottom"
    style="p3d" filename="Grids/block_2.grd" never_cut="yes">
   <boundary_surface name="imin" const_coord="z=0">
      <region range1="1:1" range2="1:-1" range3="1:-1"/>
      <boundary_condition type="symmetry"/>
   </boundary_surface>
   <boundary_surface name="jmin">
      <region range1="1:-1" range2="1:1" range3="1:-1"/>
      <boundary_condition type="solid"/>
   </boundary_surface>
   <boundary_surface name="jmax">
      <region range1="1:-1" range2="-1:-1" range3="1:-1"/>
      <boundary_condition type="farfield"/>
   </boundary_surface>
</volume_grid>
<volume_grid name="top"
    style="p3d" filename="Grids/block_3.grd" never_cut="yes">
   <boundary_surface name="imin" const_coord="z=0">
      <region range1="1:1" range2="1:-1" range3="1:-1"/>
      <boundary_condition type="symmetry"/>
   </boundary_surface>
   <boundary_surface name="jmin">
      <region range1="1:-1" range2="1:1" range3="1:-1"/>
      <boundary_condition type="solid"/>
   </boundary_surface>
   <boundary_surface name="jmax">
      <region range1="1:-1" range2="-1:-1" range3="1:-1"/>
      <boundary_condition type="farfield"/>
   </boundary_surface>
</volume_grid>
<volume_grid name="collar-interface"
    style="p3d" filename="Grids/collar-interface2.p3du">
   <boundary_surface name="imin" const_coord="z=0">
      <region range1="1:1" range2="1:-1" range3="1:-1"/>
```

```
            <boundary_condition type="symmetry"/>
          </boundary_surface>
          <boundary_surface name="imax" const_coord="z=0">
            <region range1="-1:-1" range2="1:-1" range3="1:-1"/>
            <boundary_condition type="symmetry"/>
          </boundary_surface>
        </volume_grid>
</body>
<body name="sting">
    <!--add dynamic so that the capsule and sting
        are in different dynamic groups and hence separate
        cutter surfaces will be output-->
   <dynamic/>
   <volume_grid name="sting"
          style="p3d" filename="Grids/sting-block-5.grd">
      <boundary_surface name="kmin" const_coord="z=0">
         <region range1="1:-1" range2="1:-1" range3="1:1"/>
         <boundary_condition type="symmetry"/>
      </boundary_surface>
      <boundary_surface name="kmax" const_coord="z=0">
         <region range1="1:-1" range2="1:-1" range3="-1:-1"/>
         <boundary_condition type="symmetry"/>
      </boundary_surface>
      <boundary_surface name="jmin">
         <region range1="1:-1" range2="1:1" range3="1:-1"/>
         <boundary_condition type="solid"/>
      </boundary_surface>
      <boundary_surface name="imax">
         <region range1="-1:-1" range2="1:-1" range3="1:-1"/>
         <boundary_condition type="farfield"/>
      </boundary_surface>
   </volume_grid>
   <volume_grid name="collar"
       style="p3d" filename="Grids/collar.p3du" never_cut="yes">
      <boundary_surface name="kmin-collar-sting">
         <region range1="1:-1" range2="1:33" range3="1:1"/>
         <boundary_condition type="solid"/>
      </boundary_surface>
      <boundary_surface name="kmin-collar-capsule">
         <region range1="1:-1" range2="33:-1" range3="1:1"/>
         <boundary_condition type="solid"/>
         <collar body="capsule"/>
      </boundary_surface>
```

```
            <boundary_surface name="imin" const_coord="z=0">
               <region range1="1:1" range2="1:-1" range3="1:-1"/>
               <boundary_condition type="symmetry"/>
            </boundary_surface>
            <boundary_surface name="imax" const_coord="z=0">
               <region range1="-1:-1" range2="1:-1" range3="1:-1"/>
               <boundary_condition type="symmetry"/>
            </boundary_surface>
         </volume_grid>
         <volume_grid name="sting-interface"
             style="p3d" filename="Grids/sting_cyl-interface2.p3du">
            <boundary_surface name="jmin" const_coord="z=0">
               <region range1="1:-1" range2="1:1" range3="1:-1"/>
               <boundary_condition type="symmetry"/>
            </boundary_surface>
         </volume_grid>
         <structured_cutter_surfaces filename="Grids/cap.mbxyz"/>
         <structured_cutter_surfaces filename="Grids/root.mbxyz"/>
        </body>
      </body>
      </global>
```

Figure 8-21 SUGGAR Input File for Capsule Example

Step 3: **Run** SUGGAR.

Action: At the command line, type:

```
surfasm -allow_dynamic_surface_overlap Input/Input.xml

suggar_3d_opt.linux -allow_dynamic_surface_overlap -
surface_assem donors.xml Input/Input.xml
```

Result: After these steps, there remained only one orphan, which was the result of a poor quality donor (i.e. one of the donor cells was itself a fringe cell). To resolve this, the donor quality tolerance was lowered to 0.8 (at the top of the SUGGAR input file provided in Figure 8-21), resulting in an assembly with no orphans. Once this was accomplished, the overlap minimization was enabled (using the

minimize_overlap element near the top of the input file provided in Figure 8-21), and SUGGAR was run one last time, still resulting in an assembly with no orphans. A slice along the symmetry plane of the final grid assembly is shown in Figure 8-22.

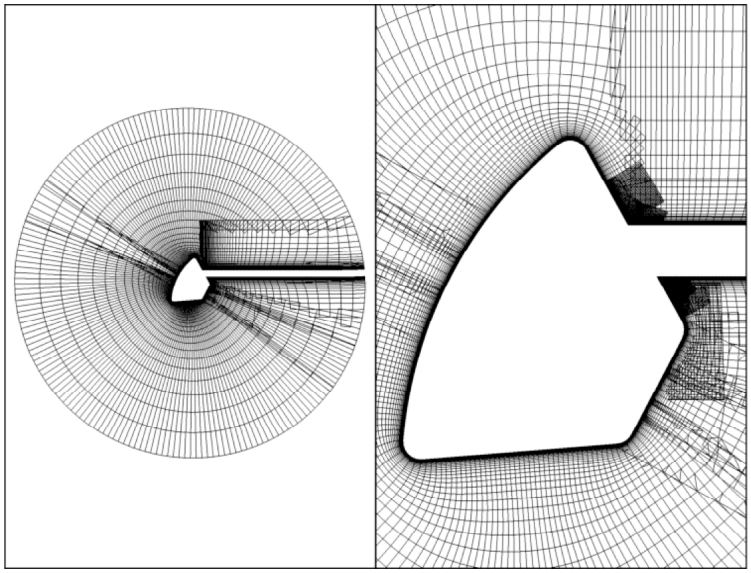


Figure 8-22 Slice Along Symmetry Plane of Overset Grid System

Step 4: **Run** FCONVERT (with `iaction=1`, `idim=3`, `inform=2`, `ouform=11`) to convert `SUGGAR/allgrids.p3dudl` to an FXDR-formatted grid file named `dplr.pgrx` suitable for parallel execution on 24 processors. For the decomposition, `kbrk` was 8, 2, and 3 in blocks 1, 4, and 5, respectively, and `ibrk` was 8 for block 6.

Step 5: **Prepare** the DPLR input file. The same operating conditions are used as in the previous two examples, i.e. molecular nitrogen as a perfect gas at M_∞= 2.0, T_∞ = 219 K, and a density corresponding to p_∞ = 2891 Pa. The maximum CFL number was set to 1000. The boundary condition flag on the overset boundaries was set to 901. The overset logic was activated by setting `iover=1` and specifying the name of the DCI file as `SUGGAR/gen_dirt.dci`.

Step 6: **Run** DPLR. In this case, DPLR was executed with the command

```
mpirun -np 24 dplr3d < dplr.inp
```

Over 2000 iterations, the RMS residual dropped about 5 orders of magnitude.

Step 7: **Run** POSTFLOW (with `ouform=25`, `interp=11`, `ivarp=110,154`) and `merge_dplr`, following the procedure outlined in Section 8.6.1. A slice along the symmetry plane of the resulting Mach number contours is shown in the left and center panels of Figure 8-23, and pressure contours on the heat shield are shown in right-most panel.

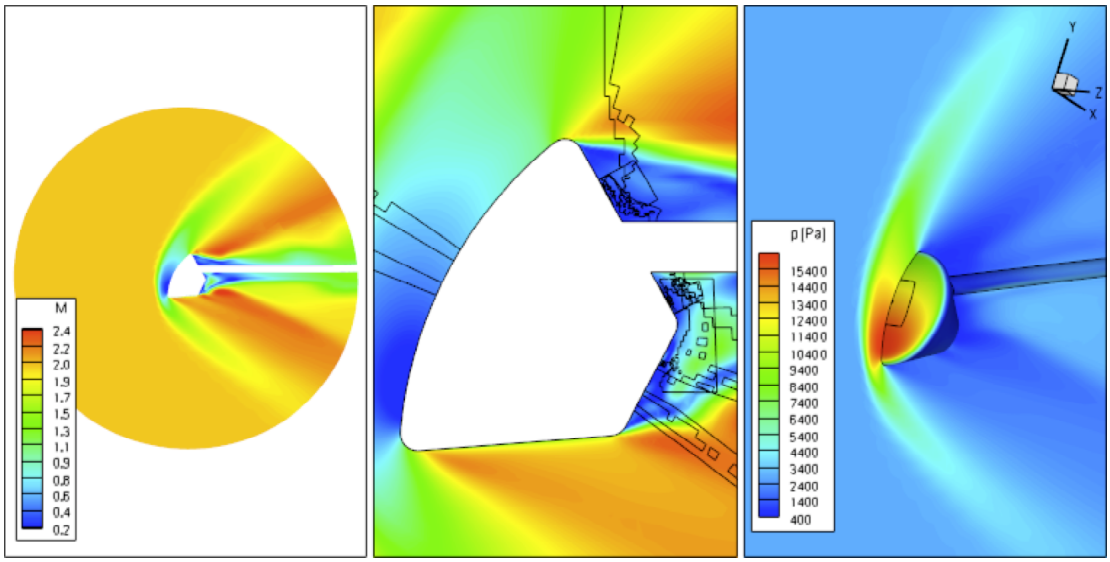


Figure 8-23 Flow Solution for Sting-Mounted Capsule Example

8.8 References

1. Noack, R.W., "DiRTlib: A Library to Add an Overset Capability to Your Flow Solver," AIAA-2005-5116, 17th AIAA Computational Fluid Dynamics Conference, Toronto, Ontario, Canada, 6-9 June 2005.

2. Boger, D.A., and Dreyer, J.J, "Prediction of Hydrodynamic Forces and Moments for Underwater Vehicles Using Overset Grids," AIAA-2006-1148, 44th AIAA Aerospace Sciences Meeting and Exhibit, Reno, Nevada, 9-12 January 2006.

3. Chan, W.B., Gomez, R.J., Rogers, S.E., and Buning, P.G., "Best Practices in Overset Grid Generation," AIAA-2002-3191, 32nd AIAA Fluid Dynamics Conference, 24-26 June 2002.

4. Noack, R.W., "SUGGAR: a General Capability for Moving Body Overset Grid Assembly," AIAA-2005-5117, 17th AIAA Computational Fluid Dynamics Conference, Toronto, Ontario, Canada, 6-9 June 2005.

5. Mehta, R.C., "Numerical Simulation of Supersonic Flow Past Reentry Capsules," Shock Waves, Vol. 15, No. 1, pp. 31-41, 2006.

Chapter 9 – Appendices

Contents

Appendices

9.0 Introduction

This section of the User Manual contains some reference material and more detailed discussions of content found in previous sections of the publication. As the DPLR Code Package is updated, additional features and reference information will be added to this section.

9.1 DPLR Code Version 4.01.1 Utilities

The following codes or scripts are provided with the DPLR package in the "utilities" directory:

- zbconvert
- dpconvert
- seqinput
- Moment
- Template

This section describes the functions and uses of each of these software tools.

9.1.1 zbconvert

zbconvert is a *Perl* script that can be used to convert zonal interface files to formats that are readable by:

- GASP® Version 3.0 (a commercially available CFD code)
- SAGe (**S**elf-**A**daptive **G**ride cod**E** – a NASA stand-alone grid-adaption application that pre-dates grid-adaption capabilities in DPLR)
- DPLR

The script is run from the command line:

```
zbconvert –i old.inter –o new.inter [-sage –dplr –gasp]
                        (-g grid.g)
```

where: `old.inter = infile` = the interface file you are converting

`new.inter = outfile` = the output file for the process

The script automatically detects the format of the input interface file and converts it to one of the supported formats specified by the `–sage`, `–dplr`, or `–gasp` flags.

Tech Tip: *If the output format is* `–sage` *(SAGe), you must also specify the associated ASCII plot3d grid file using the* `–g` *flag as shown above. This is because SAGe requires knowledge of the grid size in the input deck, and this information is not available in the interface files for either DPLR or GASP.*

9.1.2 dpconvert

dpconvert is a *Perl* script that can be used to change the format of DPLR input decks for use with different release versions of the software.

Although it is used primarily to enable rapid conversion of older DPLR input decks to a format that is compatible with the current release, it can also be used to convert a newer deck to a format that works with older versions of DPLR.

The script is run from the command line:

```
dpconvert –i old.inp –o new.inp (-V)
```

where: `old.inp = infile` = the original file you are converting

`new.inp = outfile` = the modified file

`-V` = DPLR Release Version for which file is being modified

At runtime, the script will automatically determine the version of the provided DPLR input deck '`old.inp`' and convert it to the current version. However, you can also specify a desired output version number (other than the current version), using the `–V` option.

9.1.3 seqinput

seqinput is a *Perl* script that can be used to easily sequence (coarsen) a DPLR input deck. It works by dividing the grid sizes of each block in the input deck by a specified sequencing factor.

The script is run from the command line:

```
seqinter –i old.inp –o new.inp –s I:J:K
```

where: `old.inp = infile` = the original file you are converting

`new.inp = outfile` = the sequenced file

`-s = slist` = a colon-separated list of sequencing factors in the *i-, j-,* and *k-* directions (it is assumed that all blocks are sequenced by the same factors).

At runtime, the script will generate a new DPLR input deck, and rename the input grid and restart files with the suffix "-sIJK" – a designation you can change to whatever naming convention your are using.

9.1.4 Moment

Moment is a Fortran code that generates integrated force and moment data from an input set of pointwise surface forces.

When POSTFLOW is run using `ouform=11`, POSTFLOW will automatically generate a `plot3d grid` file, a cfd function file, and a `Moment.inp` file which is the input deck for the ***Moment*** utility. Once these files have been generated, ***Moment*** is run from the command line by typing:

```
Moment < Moment.inp
```

A sample of the output from the ***Moment*** script is presented below:

```
running Moment version 3.05.0
 ----------------------------------------

 Moment Center:
   Xm =  0.000000E+00 (m)
   Ym =  0.000000E+00 (m)
   Zm =  0.000000E+00 (m)

 Reference Values:
   lref =  3.650000E+00 (m)
   aref =  4.500000E+00 (m^2)
   qdyn =  2.784862E+03 (Pa)

 Vehicle Symmetries:
   xy-plane

 Wetted Area:
   Area =  0.000000E+00 (m^2)

 Force components:
   Fx  =  1.777037E+07 (N)     ;     Cx  =  1.418013E+03
   Fy  = -1.165808E+04 (N)     ;     Cy  = -9.302740E-01
   Fz  =  0.000000E+00 (N)     ;     Cz  =  0.000000E+00

 Moment components:
   Mx  =  0.000000E+00 (N*m)   ;     Cmx =  0.000000E+00
   My  =  0.000000E+00 (N*m)   ;     Cmy =  0.000000E+00
   Mz  = -6.500303E+04 (N*m)   ;     Cmz = -1.421099E+00
```

At this time, there is no error checking in place to ensure that this output format is used correctly. So although it is not an 'error' to select other variables as output, the

results generated by the ***Moment*** utility will be incorrect unless forces per unit area are selected.

At the current time, ***Moment*** is only needed for the extraction of hinge moments because all other features of the utility are built directly into POSTFLOW.

Tech Tips:
1). *You will need to compile* ***Moment*** *as the installation script that comes with the DPLR Code Package will not automatically install the program on your system. Ask your System Administrator for information on how to compile and install this tool in your utilities directory.*

2). *Because Moment was originally written as a stand-alone tool, it has functionality that is not being used in this mode.*

9.1.5 Template

Template is a Fortran utility that can be used to automatically generate:

- zonal interface files from PLOT3D grid files
- block-specific portions of the DPLR input deck containing boundary condition information

Manually creating DPLR input and zonal interface files can be a time-consuming task. However, the ***Template*** utility, created by Scott Thomas and David Saunders and distributed with the DPLR Code Version 4.01.0 Package, can automate some or all of these two tasks depending on the grid complexity. *(Note that FCONVERT can also generate the interface file (see 'inint') but not the boundary condition portion of the input deck.)*

Overview

The name ***Template*** derives from its original intent, namely generation of *most* of the connectivity file for the multiblock flow solver FLO107MB. Block faces not adjacent to other block faces were left for their boundary conditions (e.g., subsonic outflow) to be edited into the one-line-per-block template manually.

The grid blocks were (and still are) expected to be point-to-point matched. Grids with subfacing can still be processed, but some of the interfaces will not be identified. [See use of FCONVERT for subface cases.]

The grid may contain more than one layer of blocks, but following adaptation for DPLR users, ***Template*** outputs are most complete for the common case of a single

layer of blocks. For Shuttle Orbiter applications, including local grids around damage and repair, the process has been *fully* automated with the help of ancillary input files.

Using *Template*

To generate all or most of the two DPLR control files using ***Template***, perform the following steps in the working directory containing your grid in PLOT3D multiblock form (formatted or unformatted). *(Caution: Existing* `dplr.inputs`*,* `dplr.inputs.2`*, or* `dplr.interfaces` *file will be overwritten.)*

Step 1: (Optional) Copy the '`generic.inp`' file in the `cfdinput` directory as '**`sample.inputs`**'. *(See below for ancillary input file details.)*

Step 2: (Optional) Copy the '**`template.inp.2`**' file from the `utilities` directory. *(See details below.)*

Step 3: Run TEMPLATE. (*You will be prompted for the name of the* `PLOT3D grid file` *and a tolerance to use in its detection of matching faces.)*

Step 4: Check the outputs, listed below. (*Note that some boundary conditions may need changing, while the free-stream flow conditions, CFL schedule, and flow solver iteration limit typically need editing.)*

Your working directory now contains five new files:

- `dplr.inputs` – file containing the block-specific mid-section of your DPLR input deck (boundary conditions, etc.) or possibly all of the input deck, depending on Step 1 above
- `dplr.inputs.2` – variant of `dplr.inputs` intended for possible grid sequencing
- `dplr.interfaces` – zonal interface file for the full-face interfaces of the computational grid
- `gasp.inputs` – control file in GASP flow solver format
- `template.con` – connectivity file containing (most of) the interface and BC information for the FLO107MB flow solver. Scanning this can help spot possible problems caused by block faces that don't meet the matching tolerance, i.e., look for too many integer 0s in the one-line-per-block output

Ancillary Input Files

Template looks for two control files, both of which are optional. *If either is present in the working directory, it will be invoked, so beware of unintended usage.*

If '**sample.inputs**' is present, the header and trailing portions of the sample input deck are transcribed to `dplr.inputs` and `dplr.inputs.2`. Otherwise, only the middle sections of those input decks will be produced.

If '**template.inp.2**' is present, it can serve either of two purposes, or both. Initially implemented to control the contents of `dplr.inputs.2`, it can also be used to make the automation of boundary conditions complete for specialized grids of the type developed for rapid analysis of Shuttle Orbiter damage and repair configurations. *(The latter use of `template.inp.2` is typically confined to workgroups within NASA.)*

A sample `template.inp.2` file to be used with a wing leading edge plug (or tile gap-filler) grid is shown below:

```
Sequencing controls
4 4 2
Plug blocks ! BC 2 at jmin will be changed to BC 26 (wall)
10:13
```

These controls are entered as line pairs (a text line followed by an integer list). Each line pair is optional, case-insensitive, and the order does not matter, meaning either of the pairs may be entered first or omitted.

The default grid sequencing is `2 2 1`, meaning the grid block cell counts in output file `dplr.inputs.2` are halved in the *i* and *j* directions (only), whereas the `4 4 2` shown would enable solution with the grid coarsened twice as much.

Keywords implemented for the second type of input are **Cavity** and **Plug**, with **Plug** also being appropriate for protruding tile gap filler cases. These controls allow the appropriate faces of the indicated blocks to be marked as walls (specifically, BC 26, meaning catalytic radiative equilibrium). Any reasonable format for the list of block numbers is acceptable as long as they are all on one line.

Tech Tips:

1). *You will need to compile **Template** manually as the installation script will not automatically install it on your system. Check with your System Administrator for the system-specific steps needed to compile and install this tool into your utilities directory.*

2). Template *detects matching faces by comparing the maxima and minima in x, y, and z. If two faces are found to satisfy the six possible comparisons to within the tolerance provided at run time (default epsilon = 0.0001 distance units), and the face dimensions match, then a match at all points of the face pair is likely, but the fraction of face cells for which this is true is also calculated and printed in the last column of **template.con**. Values less than 1.0 in that last column are a likely sign of **gaps** or **overlaps** in the grid.*

3). Symmetry boundary conditions *are considered only after matching block faces are checked for first. A somewhat looser tolerance is employed, namely min (10 x epsilon,*

0.001), for measuring the distance of the three pairs of coordinate maxima/minima from zero. These tests can still be fooled by (say) a flat plate in the z = 0 plane, or an almost-flat surface at x = 0. False BC entries of 17, 18, or 19 (meaning symmetry plane in x, y, or z, respectively) should be checked for under such circumstances.

***4). Template** errs in favor of **Shuttle Orbiter grids** for remaining unassigned block faces. These grids are known to contain a single layer of blocks with index k in the radial direction. A face not already assigned a flow-through BC (20) or symmetry BC (17-19) is marked as BC 26 if it is face 5 (k = 1, catalytic radiative equilibrium wall), else it is marked as BC 1 if it is face 6 (k = nk, free stream). For non-Shuttle applications, different wall BCs may need to be entered in place of BC 26.*

*Any remaining unassigned face is marked as BC 2 (specified inflow or supersonic outflow). This choice is appropriate for **local damage/repair grids** that are outside any sonic bubble. BC 2 should also be adequate for the supersonic outflow faces of ordinary grids, although occasional anomalies have been observed in baseline Shuttle solutions, so substituting BC 3 for BC 2 is recommended for such known outflow faces.*

9.2 Supported Input / Output File Formats

The DPLR Code Package Version 4.01.1 reads and/or creates the following six file types:

- Grid files, defining the discretized computational geometry of the problem.
- Zonal Interface files, describing how the blocks in multi-block grids abut each other in computational space.
- Restart (or cfd function) files, saved periodically by the CFD code to be used to restart a problem and/or post-process the solution.
- Radiation files, enabling loose coupling between DPLR and flowfield radiation analysis tools such as RADEQUIL, NEQAIR, and HARA.
- Boundary Condition files, specifying various types of pointwise boundary conditions and/or TPS material maps.
- Data files, generated by POSTFLOW for use in post-processing and data analysis of the solution.

For a more detailed discussion of each of the file types, see Chapter 6 in this User Manual.

The above-listed file types can exist in different formats. File formats supported by the DPLR Code Package are listed in the table below. Note that each supported format is assigned a unique number and a suffix which is common across the entire code package.

Table 9.1 File Formats Supported in the DPLR Code Package

Format	Description	File Type	Suffix
1	Unformatted Parallel	grid	pgrd
		restart	psln
		BC	pbcf
		radiation	prdf
11	XDR Parallel	grid	pgrx
		restart	pslx
		BC	pbcx
		radiation	prdx
21	ASCII Parallel	grid	pgra
		restart	psla
		BC	pbca
		radiation	prda
2	Unformatted Plot3D	grid	gu
		flow	qu
12	XDR Plot3D	grid	gx
		flow	qx
22	ASCII Plot3D	grid	g
		flow	q
32	Gzipped ASCII Plot3D	grid	gz
		flow	qz
3	Unformatted Plot3D	grid	gu
		flow	fu
13	XDR Plot3D	grid	gx
		flow	fx
23	ASCII Plot3D	grid	g
		flow	f
33	Gzipped ASCII Plot3D	grid	gz
		flow	fz
5	Binary Tecplot Block		plt
25	ASCII Tecplot Block		dat
6	Binary Tecplot Point		plt
26	ASCII Tecplot Point		dat

9.2.1 Format Numbers

The first digit, if any, of the file format number specifies the data-storage type as follows:

0 Written as machine-specific unformatted files. This type of file should be avoided if portability is desired, because an unformatted file created by one machine type usually cannot be read by another.

1 Written in XDR format. XDR files are binary, written to be read on any machine, and the recommended storage type for large files, including grid and restart files. *(See Tech Tip #1.)*

2 Written as an ASCII file. ASCII files are much larger than binary files, and should be avoided when possible. However, ASCII plot3d files are frequently used for grid input because they are portable and can be written by most commercial grid generation packages.

3 Written as a gzipped ASCII file. This format is currently used only for output of plot3d data from POSTFLOW.

The second digit, if any, of the file format number indicates the type of file as follows:

1 Parallel archival I/O file for use with DPLR. This is the preferred file type for grid, restart, radiation, and boundary condition files that are to be read by DPLR.

2 Plot3d grid or q-file.

3 Plot3d grid or function file. *(See Tech Tip #2)*

4 Parallel multi-file grid or restart file. *(Note: This file type is no longer supported by DPLR.)*

5 TECPLOT block file.

6 TECPLOT point file. *(See Tech Tip #3)*

Tech Tips:

1). *To read or write XDR files, the fxdr libraries must be installed on your computer and linked to DPLR during compilation. See Section 2.2 for more information.*

2). *Plot3d files cannot be read or written by DPLR2D or DPLR3D, but are frequently used to import data from or export data to other programs*

3). *TECPLOT data files are output by POSTFLOW for post-processing purposes, but cannot be read as input by any of the codes in this package. In order to create binary TECPLOT files, the TECPLOT I/O library must be properly installed and linked to DPLR. See Section 2.2 for more information.*

It is important to understand that DPLR2D and DPLR3D are separate codes, and even though many common subroutines are shared, each code requires properly dimensioned input. A common misconception is that DPLR2D reads a three-dimensional grid file, with the third dimension set to 1 and all *z*-coordinates set to zero. This is not the case. When you prepare a plot3d grid for solution by DPLR2D, your grid must be in 2D format. If a three-dimensional grid is read as input to FCONVERT with `idim=2`, the results will be unpredictable and probably not what you intended.

9.3 Parallel Decomposition

This Appendix offers a detailed discussion, with several examples, of the parallel decomposition process performed by FCONVERT on computational grids submitted to the DPLR Code Package for processing.

DPLR is a distributed-memory parallel code, so all blocks in a computational grid are computed simultaneously rather than sequentially. Multi-block information transfer is handled through MPI data constructs, so simulations must be run on at least as many processors as there are master blocks in the original computational grid.

Because running on more processors than master grid blocks is often advantageous in terms of solution speed, large blocks can be split (decomposed) into smaller pieces to increase computational efficiency and decrease turnaround time. This decomposition, if required, is performed using FCONVERT.

Although the "ideal" number of processors to use for a given job is sometimes a matter of personal preference, it is often a function of the total number of processors that are available and the number that are necessary to achieve a reasonable load balance. Once the desired number of processors to use during the run has been selected, the input grid file must be decomposed into one block per processor. This is accomplished by setting `iaction=1` or `2` in the FCONVERT input deck.

9.3.1 Load Imbalance

One of the primary metrics by which the quality of a parallel decomposition is judged is the amount of load imbalance that results. In FCONVERT, this load imbalance is computed as a measure of the average amount of wasted CPU time, assuming that the total CPU time is directly proportional to the number of grid points on a given processor. The total load imbalance (I_{tot}) is then given by:

$$I_{tot} = \sum_{n=1}^{nb} \left(N_{\max} - N_n \right) / \left(N_{\max} \cdot nb \right)$$

where *nb* is the total number of parallel blocks, N_n is the size of block *n*, and N_{max} is the size of the largest block. In practice, things are more complex than this. The type of boundary condition on each face, the number of zonal interfaces, and the relative speed of each processor all contribute to the amount of time spent on a given decomposed block in DPLR. However, the load imbalance metric is sufficient to provide a first-order estimate. The estimated total load imbalance is always reported by FCONVERT whenever a grid file is processed.

9.3.2 Decomposition Strategies

When `iaction=1`, you manually specify how each block in the input file is to be decomposed using the `ibrk, jbrk,` and `kbrk` decomposition factors. One set of decomposition factors is required for each master block in the input file. A decomposition factor of *n* implies that the block should be broken *n* times in that direction. For example, a decomposition record of:

```
Decomposition information for each master block
              ibrk    jbrk    kbrk
                2       3       1
```

indicates that the original block should be split into six by breaking it into two equal pieces in the *i*-direction and into three equal pieces in the *j*-direction. If the number of computational cells in a given direction is not evenly divisible by the selected decomposition factor, the remainder will be evenly distributed among the blocks.

Setting `iaction=1` allows you to control the way that the problem is decomposed for parallel execution, which can have significant advantages.

When `iaction=2`, you simply specifies the desired number of output blocks using the `nbreak` flag and allow FCONVERT to determine a parallel decomposition strategy that divides the original file into `nbreak` output blocks. The blocks will be broken such that load balance is maximized. This means that FCONVERT will attempt to make all blocks as close as possible to the same size. In addition, FCONVERT will attempt to make the blocks as close to cubes as possible by breaking first in the direction(s) with the most points. To do this, however, FCONVERT requires a valid DPLR input deck to exist- one that can be used to determine the locations of body surfaces in the grid file. It is a runtime error to set `iaction=2` unless a valid DPLR input deck has been specified as input.

For all decomposition strategies, it is important to minimize, and preferably eliminate, breaking the grid in the body-normal direction because DPLR is, by default, a line relaxation code that solves the Navier-Stokes equations through a series of block-

tridiagonal matrix factorizations. This method converges most rapidly when the problem has not been decomposed in the body-normal direction.

Example #1. Consider an input grid file that consists of two blocks. The plot3d header record for this case is:

```
2
17  33  129
65  65  129
```

Block #1 consists of 65,536 grid cells (16 × 32 × 128), while block #2 consists of 524,288 cells (64 × 64 × 128). Assuming that `iaction=2` and `nbreak=7` and there are no solid walls specified in the DPLR input deck, a portion of the descriptive output for this run will be:

```
Input Block 1 size: il = 16; jl =  32; kl = 128 ( 65536 cells)
Input Block 2 size: il = 64; jl =  64; kl = 128 (524288 cells)

Largest block is:
      nb =   2; original block =   2
      il =  64; jl =  64; kl =  128

Read input interface file neptune.inter
Found   3 valid zonal interface blocks in   2 block grid file

Decomposing block 1 into   1:   ibrk=  1 jbrk=  1 kbrk=  1
Decomposing block 2 into   6:   ibrk=  2 jbrk=  1 kbrk=  3
 ----------------------------------------
                  creating   7 total blocks

   7 Blocks;  Total load imbalance =  4.32%

Output Block1 size: il = 16; jl = 32; kl = 128  ( 65536 cells)
Output Block 2 size: il = 32; jl = 64; kl =  43 ( 88064 cells)
Output Block 3 size: il = 32; jl = 64; kl =  43 ( 88064 cells)
Output Block 4 size: il = 32; jl = 64; kl =  43 ( 88064 cells)
```

```
Output Block 5 size: il = 32; jl = 64; kl =  43 ( 88064 cells)
Output Block 6 size: il = 32; jl = 64; kl =  42 ( 86016 cells)
Output Block 7 size: il = 32; jl = 64; kl =  42 ( 86016 cells)
```

In this example, FCONVERT decomposed master block #2 into six nearly equal pieces while leaving block #1 unaltered. The resulting load imbalance was 4.32%.

This is the most load-balanced solution for `nbreak=7`, but it may not be the most desirable way to split the problem. For example, if the *k*-direction is body-normal for this problem, it would be preferable to select a decomposition that does not break the problem in the *k*-direction. This can be accomplished by setting `iaction=2` and specifying the correct boundary conditions in the DPLR input deck.

A portion of the FCONVERT output for this run will be:

```
Input Block 1 size: il = 16; jl =  32; kl = 128 ( 65536 cells)
Input Block 2 size: il = 64; jl =  64; kl = 128 (524288 cells)

Largest block is:
      nb = 2; original block = 2
      il =  64; jl =  64; kl =  128

 Read input interface file neptune.inter
 Found 3 valid zonal interface blocks in 2 block grid file

 Decomposing block 1 into 1:   ibrk=  1 jbrk=  1 kbrk=  1
 Decomposing block 2 into 6:   ibrk=  3 jbrk=  2 kbrk=  1
 ----------------------------------------
                  creating   7 total blocks

   7 Blocks;  Total load imbalance =  6.49%

Output Block 1 size: il = 16; jl = 32; kl = 128 ( 65536 cells)
Output Block 2 size: il = 22; jl = 32; kl = 128 ( 90112 cells)
```

```
Output Block 3 size: il = 21; jl = 32; kl = 128 ( 86016 cells)
Output Block 4 size: il = 21; jl = 32; kl = 128 ( 86016 cells)
Output Block 5 size: il = 22; jl = 32; kl = 128 ( 90112 cells)
Output Block 6 size: il = 21; jl = 32; kl = 128 ( 86016 cells)
Output Block 7 size: il = 21; jl = 32; kl = 128 ( 86016 cells)
```

The load imbalance for this case is slightly larger (6.49% vs. 4.32%), but the increased performance of the implicit algorithm would far outweigh the increase in load imbalance. Alternatively, this outcome can be accomplished by setting `iaction=1` and using the block decomposition flags to specify the desired decomposition. For this example, the following decomposition would give output identical to that obtained by using `iaction=2`:

```
Decomposition information for each master block
ibrk    jbrk    kbrk
  1       1       1
  3       2       1
```

Note that this solution is not unique; there are several other possible decompositions that would achieve the same result. The sample output for this case would be identical to that shown in the previous example.

The choice of using `iaction=1` or `2` is really dependent on the situation. For example, `iaction=1` can be used prior to generation of the DPLR input deck. In addition, `iaction=1` gives you more direct control over the decomposition performed. Because it is preferable, for the sake of efficiency, to decompose the grid so that the generation of additional zonal interfaces is minimized, using `iaction=1` and manually specifying the decomposition strategy can help you meet this condition.

For example, in the previous test problem, a decomposition strategy of:

```
Decomposition information for each master block
ibrk    jbrk    kbrk
  1       1       1
  6       1       1
```

would result in the following output:

```
Decomposing block   1 into  1:   ibrk=  1 jbrk=  1 kbrk=  1
```

```
Decomposing block   2 into  6:   ibrk=  6 jbrk=  1 kbrk=  1
 ----------------------------------------
                  creating   7 total blocks

   7 Blocks;  Total load imbalance =  6.49%

Output Block 1 size: il = 16; jl = 32; kl = 128 ( 65536 cells)
Output Block 2 size: il = 11; jl = 64; kl = 128 ( 90112 cells)
Output Block 3 size: il = 11; jl = 64; kl = 128 ( 90112 cells)
Output Block 4 size: il = 11; jl = 64; kl = 128 ( 90112 cells)
Output Block 5 size: il = 11; jl = 64; kl = 128 ( 90112 cells)
Output Block 6 size: il = 10; jl = 64; kl = 128 ( 81920 cells)
Output Block 7 size: il = 10; jl = 64; kl = 128 ( 81920 cells)
```

As you can see, this decomposition strategy results in the same load imbalance, but offers potentially improved performance because fewer additional zonal boundaries are created.

Finally, when `iaction` = 10, FCONVERT will generate an output file with the same number of blocks as the input file; i.e. no further decomposition will be performed. The same result could be achieved either by:

1) setting `iaction=1` and all `ibrk, jbrk, kbrk` flags `=1`

or

2) setting `iaction=2` and `nbreak` equal to `nborig`

FCONVERT will automatically compute all additional face, edge, and corner zonal interfaces created by the specified parallel decomposition. In addition, if the input grid contains one or more zonal interfaces, these will be automatically decomposed along with the grid file. This information will be written to the output grid file header if one of the parallel formats is requested. You can request that the resulting zonal interface file be output for informational purposes by setting `ouint= 1, 11`, or `12`.

Decomposing a file in multiple directions can create a large number of output zonal interfaces, particularly when edge and corner interfaces are considered. Because each zonal interface represents a message that must be constructed and sent via MPI send and receive calls each iteration during the CFD solution, it is generally a good idea to keep decompositions as simple as possible.

For example, if you want to run a single block 3D problem on eight processors, the simplest decomposition would be to break the problem into eight blocks in a single coordinate direction, which would generate 7 face interfaces and zero edge or corner interfaces. An alternate strategy would be to break into 4 × 2 × 1 blocks, which would generate 10 face interfaces and 6 edge interfaces, for a total of 16. The most complex decomposition would be 2 × 2 × 2 blocks. This strategy would generate 12 face interfaces, 12 edge interfaces, and 4 corner interfaces, for a total of 28. Although each of these strategies are allowed, the first would generate the least message-passing traffic during run-time, and would likely result in the most time-efficient solution.

9.3.3 Physical (Master) Blocks vs Virtual (Parllel) Blocks

The action taken by FCONVERT during a grid file decomposition depends on the output file format you specify.

If you select a plot3d output format, the input file will be physically split into multiple blocks and written as a multi-block file. If you select a parallel output file format, the input file will be "virtually split" into a number of blocks for parallel processing, but resultant file will retain information about the original physical block structure.

FCONVERT distinguishes between "virtual" blocks, which are generated purely to facilitate parallel execution, and "physical" or "master" blocks, which are a fundamental property of the input grid. Keep in mind, however, that the user interfaces to DPLR2D, DPLR3D, and POSTFLOW deal only with master blocks, and that "virtual" blocks are automatically converted to and from physical blocks as required during program execution. Therefore, when setting up a problem to run in DPLR, only the master block structure of the problem is important. If a two-block grid is decomposed into `nblk` "virtual" blocks to run in parallel, the problem is set up for DPLR as a two-block problem, regardless of the actual value of `nblk`. This means that boundary conditions, numerical models, etc. are only specified for the two master blocks. DPLR will automatically convert this information to the "virtual" values at runtime. Similarly, when the solution is post-processed by POSTFLOW, it is treated as a two-block problem, regardless of the actual number of processors that were used. This strategy greatly simplifies the preparation, execution, and post-processing overhead required for parallel jobs.

The output of FCONVERT provides information on the physical block structure, and includes physical block sizes, which are required for setting up the DPLR input deck. A portion of the output from FCONVERT for the sample problem of the previous section is shown below:

```
Summary (grid dimensions for CFD input deck):
 Hardwired to run on 7 processors

      Block   1; nx =  16; ny =  32; nz = 128
      Block   2; nx =  64; ny =  64; nz = 128
```

The summary information states that the problem has been decomposed (or "hardwired") for execution on seven processors, but there are only two physical (master) blocks that must be considered during the problem setup. Using this strategy, once a DPLR or POSTFLOW input deck has been created for a given problem, the *same* input deck can be used regardless of actual the number of processors employed in the solution. This means that you are not required to visualize or work with the parallel decomposition of the problem except when running FCONVERT.

If you select a parallel archival output format for the decomposed file (`ouform = 1, 11, 21`), a single output file will be created. This type of file actually contains only as many master blocks as specified in the original input grid file, but additional information is written to the file header to tell DPLR how to perform the appropriate decomposition at run-time. This "virtual" decomposition information is written only to the grid file header. Therefore, parallel archival restart files never need to be decomposed or recomposed. Once a parallel archival grid file has been created, it is considered to be "hardwired" for a given number of processors. This will be reflected in the output messages produced when FCONVERT is run. If you want to run or restart the problem on a different number of processors, the grid file can simply be decomposed again. FCONVERT will strip out the header information, decompose the master blocks as desired, and write the new header information into the file. No other input file type is altered in any way by changing the number of processors in the simulation.

9.3.4 Testing for Load Balance

Although many processors may be available for a run, you should try to choose a number that maximizes load balance in order to maximize the computation efficiency of the simulation.

You can test the load balance for a series of possible decompositions with FCONVERT. Set `iaction=0` and `nbreak` to the maximum number of blocks desired. FCONVERT will then loop over all possible output block numbers from the number of input blocks to the value of `nbreak`, and output the most load balanced way to decompose into that number of output blocks.

Using the same example, if `iaction=0` and `nbreak=10`, FCONVERT will generate the following output:

```
Decomposing block 1 into 1: ibrk= 1 jbrk= 1 kbrk= 1
Decomposing block 2 into 1: ibrk= 1 jbrk= 1 kbrk= 1
  2 Blocks;  Total load imbalance =  43.75%

Decomposing block 1 into 1: ibrk= 1 jbrk= 1 kbrk= 1
Decomposing block 2 into 2: ibrk= 1 jbrk= 1 kbrk= 2
  3 Blocks;  Total load imbalance =  25.00%

Decomposing block 1 into 1: ibrk= 1 jbrk= 1 kbrk= 1
Decomposing block 2 into 3: ibrk= 1 jbrk= 1 kbrk= 3
  4 Blocks;  Total load imbalance =  16.28%

Decomposing block 1 into 1: ibrk= 1 jbrk= 1 kbrk= 1
Decomposing block 2 into 4: ibrk= 2 jbrk= 1 kbrk= 2
  5 Blocks;  Total load imbalance =  10.00%

Decomposing block 1 into 1: ibrk= 1 jbrk= 1 kbrk= 1
Decomposing block 2 into 5: ibrk= 1 jbrk= 1 kbrk= 5
  6 Blocks;  Total load imbalance =   7.69%

Decomposing block 1 into 1: ibrk= 1 jbrk= 1 kbrk= 1
Decomposing block 2 into 6: ibrk= 2 jbrk= 1 kbrk= 3
  7 Blocks;  Total load imbalance =   4.32%

Decomposing block 1 into 1: ibrk= 1 jbrk= 1 kbrk= 1
Decomposing block 2 into 7: ibrk= 1 jbrk= 1 kbrk= 7
  8 Blocks;  Total load imbalance =   5.26%

Decomposing block 1 into 1: ibrk= 1 jbrk= 1 kbrk= 1
Decomposing block 2 into 8: ibrk= 2 jbrk= 2 kbrk= 2
  9 Blocks;  Total load imbalance =   0.00%
```

```
Decomposing block 1 into 1: ibrk= 1 jbrk= 1 kbrk= 1
Decomposing block 2 into 9: ibrk= 3 jbrk= 1 kbrk= 3
 10 Blocks;  Total load imbalance =   9.99%

          Finished with Load Balance Check
```

From this output summary, you can see that a perfectly load balanced solution is possible if the problem is decomposed to run on nine processors.

9.3.5 Single Block Input Files

In general, parallel decomposition must be performed by FCONVERT. However, in the special case of a single block grid with no zonal interfaces, DPLR2D and DPLR3D can perform parallel decomposition at runtime. In this case, the input grid file can simply be converted to parallel archival format (`iaction=10`). The resulting file can be run on any number of processors without further processing by FCONVERT.

9.3.6 Parallel Recomposition

FCONVERT can also be used to "recompose" a grid file that was previously decomposed by setting `iaction=3` in the FCONVERT input deck.

This option can only be used with grid files because restart, boundary condition, and radiation files are never decomposed in the first place.

In practice, this setting is rarely used because it is unnecessary to recompose parallel archival files. As previously discussed, when the FCONVERT output file is written in one of the parallel archival formats (`ouform=1,11`, or `21`), any decomposition is virtual. This means that the file merely contains header information instructing DPLR2D or DPLR3D how to properly decompose the file at runtime, eliminating any need to actively "recompose" the file. If `iaction=3` is specified with an parallel archival file as input, FCONVERT will only strip the virtual decomposition information from the file header.

If you do set `iaction=3`, you will need to specify the number of blocks in the recomposed file with the `nborig` flag. If the input file is in plot3d format, you must also provide the input interface file (`inint=1`) containing information about how the original grid or restart file was decomposed. Although FCONVERT will recompose an input grid file, it does not recreate the zonal interface file for the

recomposed problem. Therefore, be sure to save the original zonal interface file to avoid the need to recreate it after the recompose is completed.

9.4 POSTFLOW Output Variables

A complete listing of all POSTFLOW output variables is provided in Section 5.2 of this Users Manual. This appendix provides additional, detailed information about some of these variables.

The output variables in POSTFLOW are selected via the `ivarp` integer array, where each variable is assigned a unique integer quantity. These integers are a superset of those defined in the *Plot3d* and *GASP* programs, and are expressed either as non-dimensional quantities, or in SI units. (*Note: DPLR does not support English units*).

9.4.1 Grid-Related Variables

Path Length

11 path length along grid lines in *i*-direction (si)

12 path length along grid lines in *j*-direction (sj)

13 path length along grid lines in *k*-direction (sk)

Path length is determined by computing the distance from grid point to grid point in the mesh along the selected coordinate direction. For example, if `ivarp=11`, POSTFLOW will compute the path length for each constant *i* line in the output datasets. The path length is assumed to begin at zero for *ijk* = 1 and increases for increasing index.

Body Normal Distance

21 *body normal distance (dn)

The body normal distance at a surface is defined as the distance from the cell center of the first interior cell to the face center on the surface. This is the distance used in the first-order approximations of derivatives, as well as that used to define y^+ (`ivarp=581`), and the cell Reynolds number (`ivarp=59`).

Deviation from Orthogonality

22 *deviation from orthogonality [deg.] (dev)

This is defined as the number of degrees the surface-normal grid lines deviate from perfect orthogonality. For `interp=1`, this value represents a local average interpolated to the face center. The primary use of this output variable is as a measure of overall grid quality. (*Note: Orthogonality is desired at all body surfaces, but is generally unimportant at flow-through boundaries*).

9.4.2 Mixture Transport Properties

Cell Reynolds Number

59 cell Reynolds number (Re_c)

The cell Reynolds number is defined as:

$$Re_c = \frac{(a+V)\Delta\eta}{\nu}$$

where a is the sound speed, V is the local velocity magnitude, $\Delta\eta$ is the body normal distance (`ivarp=21`), and ν is the kinematic viscosity. The cell Reynolds number is typically used as a way to judge the adequacy of the near-wall spacing in a boundary layer. $Re_c < 5$ is generally sufficient to ensure accurate heat transfer and skin friction.

9.4.3 Transport Properties

Lewis Numbers

86 laminar Lewis number (Le)

96 turbulent Lewis number (Le_t)

The Lewis number Le is defined as:

$$Le = \rho D C_p / \kappa$$

where ρ is the mixture density, D is the binary diffusion coefficient, Cp is the total specific heat at constant pressure, and κ is the thermal conductivity.

Schmidt Numbers

87 laminar Schmidt number (Sc)

97 turbulent Schmidt number (Sc_t)

The Schmidt number *Sc* is defined as:

$$Sc = \frac{\mu}{\rho D}$$

where μ is the mixture viscosity, ρ is the mixture density, and D is the binary diffusion coefficient.

Prandtl Numbers

88	laminar Prandtl number (Pr)
98	turbulent Prandtl number (Pr_t)

The Prandtl number *Pr* is defined as:

$$Pr = \mu C_p / \kappa$$

where μ is the mixture viscosity, *Cp* is the total specific heat at constant pressure, and κ is the thermal conductivity.

9.4.4 Mixture Flow Properties

Stagnation Quantities

102	stagnation mixture density (r_o)
112	stagnation pressure (p_o)
122	stagnation temperature (T_o)

Stagnation quantities (density, pressure, and temperature) are computed assuming isentropic relations, and thus are not valid for a flowfield with varying isentropic exponent (γ). The stagnation quantities are defined as:

$$p_o = pS^{\frac{\gamma}{\gamma-1}}$$

$$\rho_o = \rho S^{\frac{1}{\gamma-1}}$$

$$T_o = T^S$$

where S is the entropy, defined below.

Pressure

111 dynamic pressure (Q)

The dynamic pressure Q is simply:

$$Q = \rho V^2 / 2$$

114 pressure coefficient (C_p)

The pressure coefficient is defined as:

$$(p - p_\infty) / Q_\infty$$

where Q_∞ is the freestream dynamic pressure.

Temperature

121 bulk temperature (T_b)

The bulk temperature is defined as in AIAA Paper No. 2001-2886:

$$T_b = \frac{V^2}{2C_p}$$

Ionization

180 degree of ionization (zeta)

$$\zeta = n_e / n_t$$

9.4.5 Surface Properties

Heat Transfer

512 heat transfer coefficient in mass flux units (Chm)

This is the heat transfer coefficient expressed in kg/m^2•s for use with FIAT.

$$C_{hm} = \frac{q}{\left(h_\infty - h_w\right)}$$

520 radiative equilibrium heat transfer (Qeq)

$$q_{eq} = \varepsilon \sigma T_w^4$$

This is the surface heat transfer as computed using the radiative equilibrium wall formation. In this expression, ε is the surface emissivity, σ is the Stefan-Boltzmann constant, and T_w is the surface temperature. This variable is provided mainly as a sanity check to ensure that the computed heat transfer agrees with the radiative equilibrium value when a radiative equilibrium wall is specified.

9.5 Reference Terms

Lewis Number (Le)	$Le = \rho D C_p / \kappa$
Schmidt Number (Sc)	$Sc = \frac{\mu}{\rho D}$
Turbulent Lewis Number (LeT)	$Le_T = \rho D_T C_p / \kappa_T$
Turbulent Schmidt Number (ScT)	$Sc_T = \frac{\mu_T}{\rho D_T}$
Prandtl Number	$Pr = \mu C_p / \kappa$
Turbulent Prandtl Number	$\mathrm{Pr}_T = \rho \mu_T / \left(\kappa_T C_p \right)$
cell Reynolds Number	$\mathrm{Re}_c = \left(\frac{\rho c}{\mu} \right)_w \Delta \eta$

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
27/10/2009	Technical Memorandum	

4. TITLE AND SUBTITLE

Data Parallel Line Relaxation (DPLR) Code User Manual
Acadia - Version 4.01.1

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Dr. Michael J. Wright, Todd White, Nancy Mangini

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER
WBS 999574.01.02.01.01

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Ames Research Center, Moffett Field, CA 94035-1000

8. PERFORMING ORGANIZATION REPORT NUMBER
A-090018

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Washington, D. C. 20546-0001

10. SPONSORING/MONITOR'S ACRONYM(S)
NASA

11. SPONSORING/MONITORING REPORT NUMBER
NASA/TM–2009-215388

12. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified — Unlimited Distribution: Nonstandard
Subject Category: 02, 05, 34
Availability: NASA CASI (301) 621-0390

13. SUPPLEMENTARY NOTES
Point of Contact: Dr. Michael Wright, Ames Research Center, MS 230-2, Moffett Field, CA 94035-1000
(650) 604-4210

14. ABSTRACT

Data-Parallel Line Relaxation (DPLR) code is a computational fluid dynamic (CFD) solver that was developed at NASA Ames Research Center to help mission support teams generate high-value predictive solutions for hypersonic flow field problems. The DPLR Code Package is an MPI-based, parallel, full three-dimensional Navier-Stokes CFD solver with generalized models for finite-rate reaction kinetics, thermal and chemical non-equilibrium, accurate high-temperature transport coefficients, and ionized flow physics incorporated into the code. DPLR also includes a large selection of generalized realistic surface boundary conditions and links to enable loose coupling with external thermal protection system (TPS) material response and shock layer radiation codes.

15. SUBJECT TERMS
Hypersonic flow, Computational Fluid Dynamics, Simulations, Navier-Stokes, CFD solver, Structured grids

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Michael J. Wright
Unclassified	Unclassified	Unclassified	Unclassified	273	19b. TELEPHONE *(Include area code)* (650) 604-4210